|  |
| --- |
| **ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ** |
|  |
|  |
|  |
| **Практическое занятие №5.**  **Тема: Проектирование классов с наследованием** |
|  |
|  |
|  |

**Цель:** получение практических навыков проектирования классов и разработки механизмов открытого и закрытого наследования в С++.

**Основные теоретические положения**

**Наследование**

Наследование – это процесс, посредством которого один объект может приобретать свойства другого. Так как объекты принадлежат к классам, то между классами в этом случае устанавливаются определенные взаимоотношения. Класс, который делегирует свойства, называется базовым (base), а который наследует свойства – производным (derived). У одного базового класса может быть несколько производных, и наоборот, один производный класс может наследовать характеристики нескольких базовых классов.

Схемные примеры наследования:

**а)** В1 **б)** В1 В2 **в)** В1

D1 D1

D

D2

В примере (а) показана простейшая схема подчиненности с одним базовым (В1) и одним производным (Д1) классом. В примере (б) показан один производный (D) класс от двух базовых (В1 и В2). На схеме (в) показано, что В1 есть базовый класс для D1, который в свою очередь является базовым для D2. В этом последнем случае В1 называется для D2 косвенно-базовым. Взаимосвязи классов в программе образуют **иерархию классов**.

При указании наследования соблюдаются следующие правила.

Базовый класс описывается обычным образом. Например:

**class base**

**{**

**//содержание класса base**

**};**

Для указания наследования после имени производного класса ставится двоеточие и указывается имя базового класса со **спецификатором доступа** (private, public). Для примера (а) описание будет выглядеть следующим образом:

**class derived: public base{**

**//содержание класса derived**

**};**

Спецификатор выбирается в зависимости от того, как предполагается использовать члены класса base.

Для примера (б) описание производного класса будет выглядеть, например, так:

**class derived: public В1, public В2 {**

**//содержание класса derived**

**};**

В примере (в) для производного класса D2 базовым будет класс D1, поэтому описание D2 соответствует схеме (а).

**Взаимоотношения между членами классов и доступ к ним.**

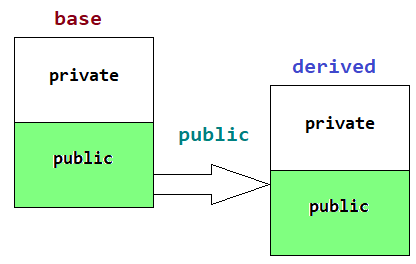
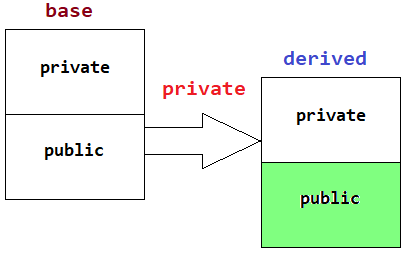
Базовый класс является первым в иерархии, он «не знает» сколько у него будет производных классов (или не будет вообще). Поэтому 1) он проектируется самодостаточным в плане доступа к закрытым членам и 2) члены производных классов в нем не упоминаются.

Как известно, **внутри** класса его члены открыты друг для друга: любая функция-член может обращаться к любой переменной-члену класса. Однако **извне** можно обратиться только к открытым членам класса.

Важно отметить, что при наследовании производный класс **полностью** включает в себя базовый класс. Однако не все члены классов открыты друг для друга в этом объединении. Здесь необходимо следовать следующим правилам.

1. «Круг общения» членов производного класса расширяется только за счет publiс-членов базового класса. Это означает, что члены производного класса могут обращаться **напрямую** только к открытым членам класса **base**; private-члены базового класса можно использовать в описании функций производного класса только через открытые члены base class.

2. Если наследование производится со спецификатором доступа **public**, то открытые члены base становятся public-членами класса derived; если наследование производится по типу **private**, то public-члены базового класса становятся private-членами derived. Private-члены базового класса **безусловно** наследуются производным классом, но только обращаться к ним члены производного класса derived могут исключительно посредством public-членов base.



Пример.

**#include <iostream>**

**using namespace std;**

**class base**

**{ int x;**

**public:**

**void set\_x (int n) {x=n;}**

**void show\_x ( ) {cout<<x<<’\n’;}**

**};**

**class derived: public base**

**{ int y;**

**public:**

**void set\_y (int k) {y=k;}**

**void show\_y ( ) {cout<<y<<’\n’;}**

**};**

**int main ( )**

**{**

**derived obj;**

**obj.set\_x(10);**

**obj.set\_y(12);**

**obj.show\_x( );**

**obj.show\_y( );**

**return 0;**

**}**

В данном примере класс base наследуется как открытый. Поэтому в объекте obj, который является экземпляром производного класса, его public- функции доступны как обычные открытые члены. Но “добраться” напрямую до закрытого члена базового класса всё также невозможно; и инструкция типа

cout <<obj.x; // Неверно!

будет ошибкой.

Теперь рассмотрим пример, в котором два класса наследуют один базовый класс по типу **public**. В базовом классе одна закрытая переменная и у каждого наследника по одной закрытой переменной. Кроме переменной в каждом классе по две public-функции для доступа к этой переменной.

**#include <iostream>**

**using namespace std;**

**class base{**

**int x;**

**public:**

**void setx(int a){x=a;}**

**int getx(){return x;}**

**};**

**class one: public base{**

**int y;**

**public:**

**void sety(int a) {y=a;}**

**int gety(){return y;}**

**};**

**class two: public base{**

**int z;**

**public:**

**void setz(int a) {z=a;}**

**int getz() { return z; }**

**};**

**int main()**

**{**

**one ob1;**

**two ob2;**

**ob1.setx(12);**

**ob1.sety(15);**

**ob2.setx(33);**

**ob2.setz(37);**

**int sum=ob1.getx()+ ob2.getx();**

**cout<< sum<<endl;**

**if(ob1.gety()>ob2.getz())**

**cout<< "Max = "<<"one"<<endl;**

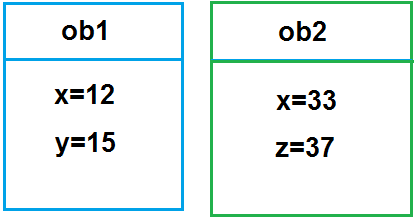
**else**

**cout<< "Max = "<<"two"<<endl;**

**return 0;**

**}**

В главной функции создается объект класса one и объект класса two.Т.к. это - производные классы, то их объекты содержат как собственные переменные, так переменную базового класса. Состав объектов ob1 и ob2 показан на рисунке.



Затем определяется сумма значений переменной **x** двух объектов и максимальное из значений переменных **y** и **z.**

На экране будет:



**Защищенные члены класса**

Из вышеизложенного все больше проступает неудобство работы с закрытыми членами базового класса в производном классе. Чтобы допустить членов производного класса к общению с private-членами базового класса, введен спецификатор доступа **protected** (защищенный).

Теперь члены класса разделены на три группы: private, protected и public. Спецификатор protected может располагаться в любом месте описания класса, но обычно он располагается перед public. И это оправдано, потому что protected-члены – это часть закрытых членов базового класса, к которым напрямую могут обращаться члены производных классов.

}

##### Члены класса

закрытые члены

protected

private

public

**Конструкторы и деструкторы в наследовании**

Если у базового и производного классов имеются конструкторы и деструкторы, то конструктор базового класса выполняется раньше конструктора производного; для деструкторов соблюдается обратный порядок. Если конструкторы имеют параметры, то:

а) все аргументы для конструктора базового и конструктора производного классов передаются конструктору производного класса;

б) затем, используя расширенную форму объявления конструктора производного класса, соответствующие аргументы передаются в базовый класс.

**#include<iostream>**

using namespace std;

**class base {**

**int i;**

**public:**

**base(int n) {cout <<”Base constructor”; i=n;}**

**~base ( ) {cout<<”Base destructor”;}**

**void show\_i( ) {cout <<i<<'\t';}**

**};**

**class derived : public base {**

**int j;**

**public:**

**derived(int n, int m): base(m)**

**{cout<<”Derived constructor”; j=n;}**

**~derived( ) {cout<<”Derived destructor”;}**

**void show\_j( ) {cout<<j<<’\n’;}**

**};**

**int main( )**

**{**

**derived obj(10,20);**

**obj.show\_i( );**

**obj.show\_j( );**

**return 0;**

**}**

Оба класса в примере имеют конструкторы и деструкторы.

Вначале на экран выведется:

Base constructor,

затем - Derived constructor.

Конструктор производного класса объявлен так, что требует два аргумента; один он использует сам, другой передает в базовый класс. В результате инициализируются переменные **i** и **j**.

Затем на экран будет выведено:

20 10.

После этого выводится сообщение:

Derived destructor,

а потом- Base destructor.

**Дружественные функции в производных классах**

Для производных классов также можно создавать дружественные функции с целью прямого доступа к закрытым членам классов. Рассмотрим особенности на примере.

**#include <iostream>**

**using namespace std;**

**class two;** //опережающее объявление производного класса

**class base{**

**int x;**

**public:**

**void setx(int a) {x=a;}**

**int getx() { return x; }**

**};**

**class one: public base{**

**int y;**

**public:**

**void sety(int a) {y=a;}**

**int gety() { return y; }**

**friend int minimum(one oob, two tob);**

**};**

**class two: public base{**

**int z;**

**public:**

**void setz(int a) {z=a;}**

**int getz() { return z; }**

**friend int minimum(one oob, two tob);**

**};**

//описание дружественной функции

**int minimum(one oob, two tob){**

**if(oob.y<tob.z)**

**{cout<< "Min = "<<"one"<<endl;**

**return oob.y;**

**}**

**else**

**{cout<< "Min = "<<"two"<<endl;**

**return tob.z;**

**}**

**}**

**int main()**

**{**

**one ob1;**

**two ob2;**

**ob1.setx(12); ob1.sety(15);**

**ob2.setx(33); ob2.setz(37);**

**cout<< minimum(ob1,ob2);**

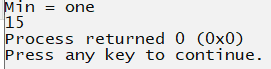
**return 0;**

**}**

В данном примере дружественная функция **friend int minimum(one oob, two tob)** предназначена для работы с объектами двух производных классов, на что указывает список её параметров. Заявленная вначале в производном классе **one** дружественная функция использует имя второго производного класса **two**, который описан ниже в программе и поэтому не известен компилятору в данной точке программы. Для указания компилятору того, чем является имя **two**, в начале программы произведено опережающее объявление класса **two**: **class two;**

Внутри дружественной функции обращение к закрытым членам классов производится непосредственно, например **if(oob.y<tob.z).**

Вид экрана после выполнения программы:



**Генерация случайных чисел**

Случайные числа в языке программирования С++ могут быть сгенерированы функцией **rand()** из стандартной библиотеки С++. Функция **rand()** генерирует числа в диапазоне от 0 до RAND\_MAX. RAND\_MAX - это константа, определённая в библиотеке <cstdlib>.

Реализация генератора случайных чисел:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

// константа верхнего предела случайных чисел

**cout << "RAND\_MAX = " << RAND\_MAX << endl;**

// запуск генератора случайных чисел

**cout << "random number = " << rand() << endl; return 0;**

**}**

Вид экрана:

****

Для генерации случайных чисел в заданном диапазоне используется инструкция:

**random\_number = firs\_value + rand() % last\_value;**

где **firs\_value** - минимальное число из желаемого диапазона, а **last\_value** - ширина выборки.

**Сборка проекта из нескольких файлов**

Для удобства работы с программой её разбивают на несколько частей (файлов). Файлы различают:

* заголовочные, которые обозначаются с расширением «**.h** »;
* исходного кода, которые обозначаются с расширением **«.cpp**».

Главная функция помещается в файл с расширением **«.cpp**». К нему в тексте программы с помощью директивы **#include** подключаются остальные файлы, например,

**#include ”Classes.h”.**

Имя файла указывается в кавычках (а не в угловых скобках), чтобы не путать с библиотечными файлами.

**Практические задания**

*Выберите один из вариантов задания.*

**Вариант А.**

**Задание 1.** Спроектировать структуру классов в соответствии с индивидуальным заданием (см. таблица 7). Наследование одного из производных классов спроектировать по типу **private**. *Вычисляемый показатель* реализовать для объектов производных классов в отдельной функции, не являющейся дружественной к классам.

**Задание 2.** Для каждого производного класса динамически выделить память для 5-элементного массива объектов. Свойства объектов задавать случайным образом. Определить *Вычисляемый показатель*.

**Задание 3.** Проект разместить в нескольких файлах.

**Вариант В.**

**Задание 1**. Спроектировать структуру классов в соответствии с индивидуальным заданием (см. таблица 7). Наследование классов осуществляется по типу **public**. Для определения *Вычисляемого показателя* спроектировать функцию для работы с объектами производных классов (не являющейся дружественной к классам).

**Задание 2**. Для каждого производного класса динамически выделить память для 5-элементного массива объектов. Свойства объектов задавать случайным образом. Определить *Вычисляемый показатель*.

**Задание 3.** Проект разместить в нескольких файлах.

**Вариант С.**

**Задание 1**. Спроектировать структуру классов в соответствии с индивидуальным заданием (см. таблица 7).

**Задание 2.** Создать несколько объектов. Рассчитать *Вычисляемый показатель*.

Отчет оформляется по общеустановленным правилам в *электронном виде* со следующим содержанием:

1. титульный лист,
2. тема и цель практической работы,
3. задание на практическую работу,
4. текст программы с комментариями,
5. результаты работы программы (вид экрана) и
6. выводы по созданному проекту и использованию средств языка программирования.

Таблица 7

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п\п** | **Базовый класс** | **Производные классы** | **Вычисляемый показатель** |
|  | Компьютер | Настольный компьютер, ноутбук | Экземпляр с наибольшей оперативной памятью |
|  | Локальная сеть | Одноранговая сеть, сеть типа клиент-сервер | Минимальная стоимость монтажа |
|  | Транспортное средство | Легковой автомобиль, грузовой автомобиль | Максимальный пробег на полном бензобаке |
|  | Программный продукт | Операционная система, текстовый редактор | Последняя версия |
|  | Документ | Паспорт, студенческий билет | Количество документов, выданных в прошлом году |
|  | Диета | Для здоровья, для похудения | Максимальное дневное количество белков |
|  | Периферийное устройство компьютера | Клавиатура, мышь | Минимальная цена устройства |
|  | Строительный товар | Инструменты, сыпучие материалы | Товар с максимальной ценой |
|  | Представитель университета | Преподаватель, студент | Количество студентов, обучающихся у конкретного преподавателя |
|  | Предприятие малого бизнеса | Магазин, парикмахерская | Название предприятия с максимальным числом сотрудников |
|  | СУБД | Реляционная, иерархическая | Количество СУБД заданного производителя |
|  | Страховой полис | Полис обязательного медицинского страхования, страхования жилища | Количество полисов на заданную фамилию |
|  | Наушники | Проводные, беспроводные | Тип с максимальным частотным диапазоном |
|  | Недвижимость | Таунхаус, квартира в многоквартирном доме | Максимальная жилая площадь |
|  | Часы | Электронные часы, механические часы | Самый дорогой экземпляр |
|  | Телефон | Смартфон, кнопочный | Самая новая модель |
|  | Бытовая техника | Телевизор, холодильник | Количество товаров заданной фирмы |
|  | Магнитная карта для проезда на транспорте | Карта общего назначения для проезда в метро, льготная транспортная карта учащегося | Количество карт без поездок |
|  | Насекомое | Стрекоза, бабочка | Максимальный размер крыльев |
|  | Канцелярские товары | Бумага, авторучка | Количество товаров заданной фирмы |
|  | Товар | Посуда, продукты питания | Сумма покупки |
|  | Учащийся | Школьник, студент | Учащийся с максимальным IQ (коэффициентом интеллекта) |
|  | Путешествие (тур) | Морской круиз, отдых на море | Самый дешевый тур на 7 и более дней |
|  | Мебель | Кровать, стол | Количество предметов из дерева |
|  | Программное обеспечение (ПО) | Системное ПО, прикладное ПО | Представители с максимальным и минимальным объемом занимаемой памяти |
|  | Представление | Театр, кино | Минимальное число зрителей в зале |
|  | Запоминающее устройство | Флэш-карта, карта памяти | Экземпляры с максимальным и минимальным размером |
|  | Принтер | Струйный, лазерный | Представитель с наибольшей производительностью |
|  | Книга | Электронная, на бумаге | Количество книг одного автора |
|  | Телефон | Смартфон, кнопочный | Самая новая модель |