**6.2. Простое наследование**

Итак, для того чтобы показать, что класс В наследует класс A (класс B выведен из класса A), в определении класса B после имени класса ставится двоеточие и затем перечисляются классы, из которых B наследует:

class A

{

public:

A();

~A();

MethodA();

};

class B : public A

{

public:

B();

. . .

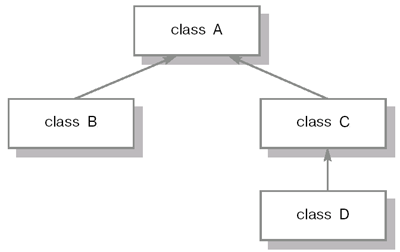
};

Термин "наследование" означает, что класс B обладает всеми свойствами класса A, он их унаследовал. У объекта *производного класса* Bесть все атрибуты и методы *базового класса* A. Разумеется, новый класс может добавить собственные атрибуты и методы.

B b;

b.MethodA(); // вызов метода базового класса

Часто выведенный класс называют подклассом, а *базовый класс* – суперклассом. Из одного *базового класса* можно вывести сколько угодно подклассов. В свою очередь, *производный класс* может служить *базовым* для других классов. Изображая отношения *наследования*, их часто рисуют в виде иерархии или дерева:



Иерархия классов может быть сколь угодно глубокой. Если нужно различить, о каком именно классе идет речь, класс C называют непосредственным или *прямым* *базовым классом* класса D, а класс A – косвенным *базовым классом* класса D.

Предположим, что для библиотечной системы необходимо создать классы, описывающие различные книги, журналы и т.п., которые хранятся в библиотеке. Книга, журнал и газета обладают как общими, так и различными свойствами. У книги имеется автор или авторы, название и год издания. У журнала есть название, номер и содержание – список статей. В то же время книги, журналы и т.д. имеют и общие свойства: все это – "единицы хранения" в библиотеке, у них есть инвентарный номер, они могут быть в читальном зале, у читателей или в фонде хранения. Их можно выдать и, соответственно, сдать в библиотеку. Эти общие свойства удобно объединить в одном *базовом классе*. Введем класс Item, который описывает единицу хранения в библиотеке:

class Item

{

long invNumber; // инвентарный номер — целое число

bool taken; // хранит состояние объекта - взят на руки

public:

Item();

~Item();

bool IsTaken(); // истина, если единица хранения на руках

bool IsAvailable(); // истина, если этот предмет имеется в библиотеке

long GetInvNumber(); // возвращает инвентарный номер

void Take(); // операция "взять"

void Return(); // операция "вернуть"

};

Когда мы разрабатываем часть системы, которая имеет дело с процессом выдачи и возврата книг, вполне достаточно того интерфейса, который представляет *базовый класс*. Например:

// выдать на руки

void TakeAnItem(Item i)

{

. . .

if (i.IsAvailable())

i.Take();

}

Конкретные свойства книги будут представлены классом Book.

class Book : public Item

{

String author; // автор

String title; // название

String publisher; // издательство

short year; // год выпуска

public:

String Author();

String Title();

String Publisher();

long YearOfPublishing();

String Reference(); // полная ссылка на книгу

};

Для журнала класс Magazine предоставляет другие сведения:

class Magazine : public Item

{

String volume;

short number;

String title;

Date date;

public:

String Volume(); // том

short Number(); // номер

String Title(); // название

Date DateOfIssue();// дата выпуска

};

У объекта класса Book имеются методы, непосредственно определенные в классе Book, и методы, определенные в классе Item.

Book b;

long in = b.GetInvNumber();

String t = b.Reference();

*Производный класс* имеет доступ к методам и атрибутам *базового класса*, объявленным во внешней и защищенной части *базового класса*, однако доступ к внутренней части *базового класса* не разрешен. Предположим, в качестве части полной ссылки на книгу решено использовать инвентарный номер. Метод Reference класса Book будет выглядеть следующим образом:

String

Book::Reference(void) const

{

String result = author + "\n"

+ title + "\n"

+ String(GetInvNumber());

return result;

}

Запись:

String result = author + "\n"

+ title + "\n"

+ String(invNumber);

не разрешена, поскольку invNumber – внутренний атрибут класса Item. Однако если бы мы поместили invNumber в защищенную часть класса:

class Item

{

. . .

protected:

long invNumber;

};

то методы классов Book и Magazine могли бы непосредственно использовать этот атрибут.

Назначение защищенной (***protected***) части класса в том и состоит, чтобы, закрыв доступ "извне" к определенным атрибутам и методам, разрешить пользоваться ими *производным классам*.

Если одно и то же имя атрибута или метода встречается как в *базовом классе*, так и в *производном*, то *производный класс* перекрывает *базовый*.