

Тема.2.8 Наследование, полиморфизм и виртуальные функции

Часть 2

Методы классов и множественное наследование

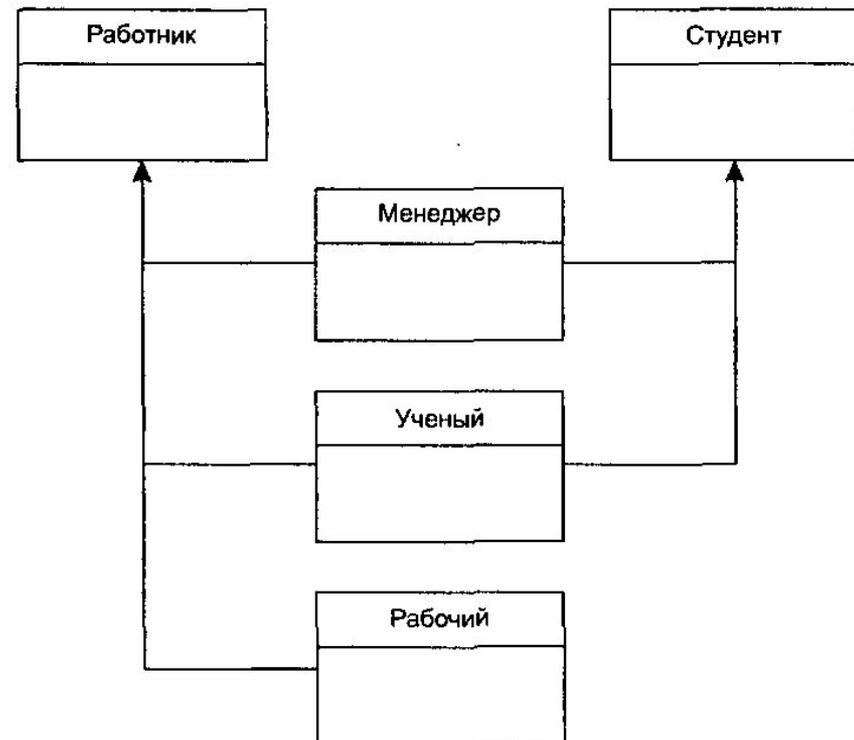
- Пусть нам для некоторых служащих необходимо указать их образование в программе EMPLOY.
- Пусть в другой программе существует класс student, в котором указывается образование каждого студента.
- Тогда вместо изменения класса employee можно воспользоваться данными класса student с помощью множественного наследования.

Методы классов и множественное наследование

- В классе `student` содержатся сведения о школе или университете, которые закончил студент, и об уровне полученного им образования.
- Эти данные хранятся в строковом формате.
- Методы `getedu()` и `putedu()` позволяют ввести данные о студенте и просмотреть их.
- Информация об образовании нужна не для всех служащих.

Методы классов и множественное наследование

- Пусть не нужны записи об образовании рабочих, а необходимы только записи об ученых и менеджерах.
- Поэтому необходимо модифицировать классы `manager` и `scientist` так, что они будут являться производными классов `employee` и `student`



Конструкторы при множественном наследовании

- Рассмотрим пример, показывающий, как работают конструкторы при множественном наследовании.
- Представим, что мы пишем программу для строителей-подрядчиков, которая работает со строительными материалами. Нам нужен класс, определяющий количество стройматериалов каждого типа, например 100 восьмиметровых бревен. Другой класс должен хранить различные данные о каждом виде стройматериалов. Например, длину куска материала, количество таких кусков и стоимость за погонный метр.
- Нам также нужен класс, хранящий описание каждого вида стройматериала, которое включает в себя две части. В первой части номинальные размеры поперечного сечения материала. Они измеряются дюймами. Во второй — сорт материала. Класс включает в себя строковые поля, которые описывают номинальные размеры и сорт материала. Методы класса получают информацию от пользователя, а затем выводят ее на экран.

Конструкторы без аргументов

- В классе Type конструктор без аргументов выглядит следующим образом: Type ()
- *{ strcpy(dimensions, "N/A"); strcpy grade, "N/A") }*
- Этот конструктор присваивает значениям полей dimensions и grade строку "N/A" (недоступно), поэтому при попытке вывести данные для объекта класса Lumber пользователь будет знать, что поля пусты.

Конструкторы без аргументов

- Есть конструктор без аргументов в классе Distance:
- *Distance () : feet (0). inches (0.0) { }*
- Конструктор без аргументов класса Lumber вызывает конструкторы обоих классов — Type и Distance.
- *Lumber : Type (). Distance (). quantity (0). price (0.0) { }*
- Имена конструкторов базового класса указаны после двоеточия и разделены запятыми.
- При вызове конструктора класса Lumber начинают работу конструкторы базовых классов Type() и Distance().
- При этом инициализируются переменные quantity и price.

Конструктор со многими аргументами

- Конструктор класса `Type` с двумя аргументами выглядит следующим образом:
- ***Type (string di. string gr) : dimensions (di). grade (gr)***
- Этот конструктор копирует строковые аргументы в поля класса `dimensions` и `grade`.

Конструктор со многими аргументами

- Конструктор класса Distance :
- *Distance (int ft. float in) : feet (ft). inches (in) { }*
- В конструктор класса Lumber включены оба этих конструктора, которые получают значения для аргументов.

```
Lumber ( string di. string gr. // параметры для Type
int ft. float in. // параметры для Distance
int qu. float pre ) : // наши собственные параметры
Type (di.gr). // вызов конструктора lype
Distance ( ft. in ). // вызов конструктора Distance
quantity ( qu ). price ( pre ) // вызов наших конструкторов
t)
```

Конструктор со многими аргументами

- Кроме того, класс `Lumber` имеет и свои аргументы: количество материала и его цена.
- Таким образом, конструктор имеет шесть аргументов.
- Он вызывает два конструктора, которые имеют по два аргумента, а затем инициализирует два собственных поля.

Неопределенность при множественном наследовании

- В определенных ситуациях могут появиться некоторые проблемы, связанные со множественным наследованием.
- Допустим, что в обоих базовых классах существуют методы с одинаковыми именами, а в производном классе метода с таким именем нет.
- Как в этом случае объект производного класса определит, какой из методов базовых классов выбрать?
- Одного имени метода недостаточно, поскольку компилятор не сможет вычислить, какой из двух методов имеется в виду.

```
// ambigu.cpp
//демонстрация неопределенности при множественном наследовании
#include <iostream>
using namespace std;
////////////////////////////////////
class A
{
public:
void show ( ) { cout << "Класс A\n"; }
};
class B {
public:
void show ( ) { cout << "Класс B\n"; }
};
class C : public A, public B {
};
////////////////////////////////////
int main ( )
{
C objC;    // объект класса C
// objC.show ( ); // так делать нельзя - программа не скомпилируется
objC.A::show ( ); // так можно
objC.A::show ( ); // так можно
return 0;
}
```

- Проблема решается путем использования оператора разрешения, определяющего класс, в котором находится метод.
- Таким образом,
- ***ObjC.A::show ()***:
- Направляет к версии метода `showQ`, принадлежащей классу `A`, а ***ObjC.B: -.show ()***:
- Направляет к методу, принадлежащему классу
- В. Б. Страуструп называет это устранением неоднозначности.

- Другой вид неопределенности появляется, если создать производный класс от двух базовых классов, которые, в свою очередь, являются производными одного класса.
- Это создает дерево наследования в форме ромба.

```
// diamond.cpp
// демонстрация наследования в форме ромба
#include <iostream>
using namespace std;
////////////////////////////////////
class A
{
public:
void func ( );
};
class B : public A { };
class C : public A { };
class D : public B, public C { };


---


////////////////////////////////////
int main ( )
{
D ObjD;
//ObjD.func ( ); // неоднозначность: программа не скомпилируется
return 0;
}
```

- Классы В и С являются производными класса А, а класс D является производным классов В и С.
- Трудности начинаются, когда объект класса D пытается воспользоваться методом класса А.
- В примере выше объект objD использует метод func().
- Однако классы В и С содержат в себе копии метода func(), унаследованные от класса А.
- Компилятор не может решить, какой из методов использовать, и сообщает об ошибке.