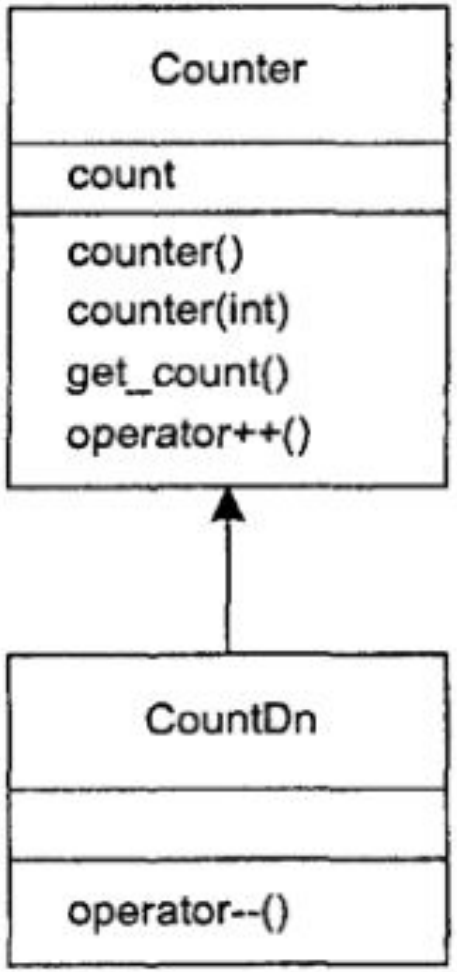
Объектно-ориентированное программирование

Наследование

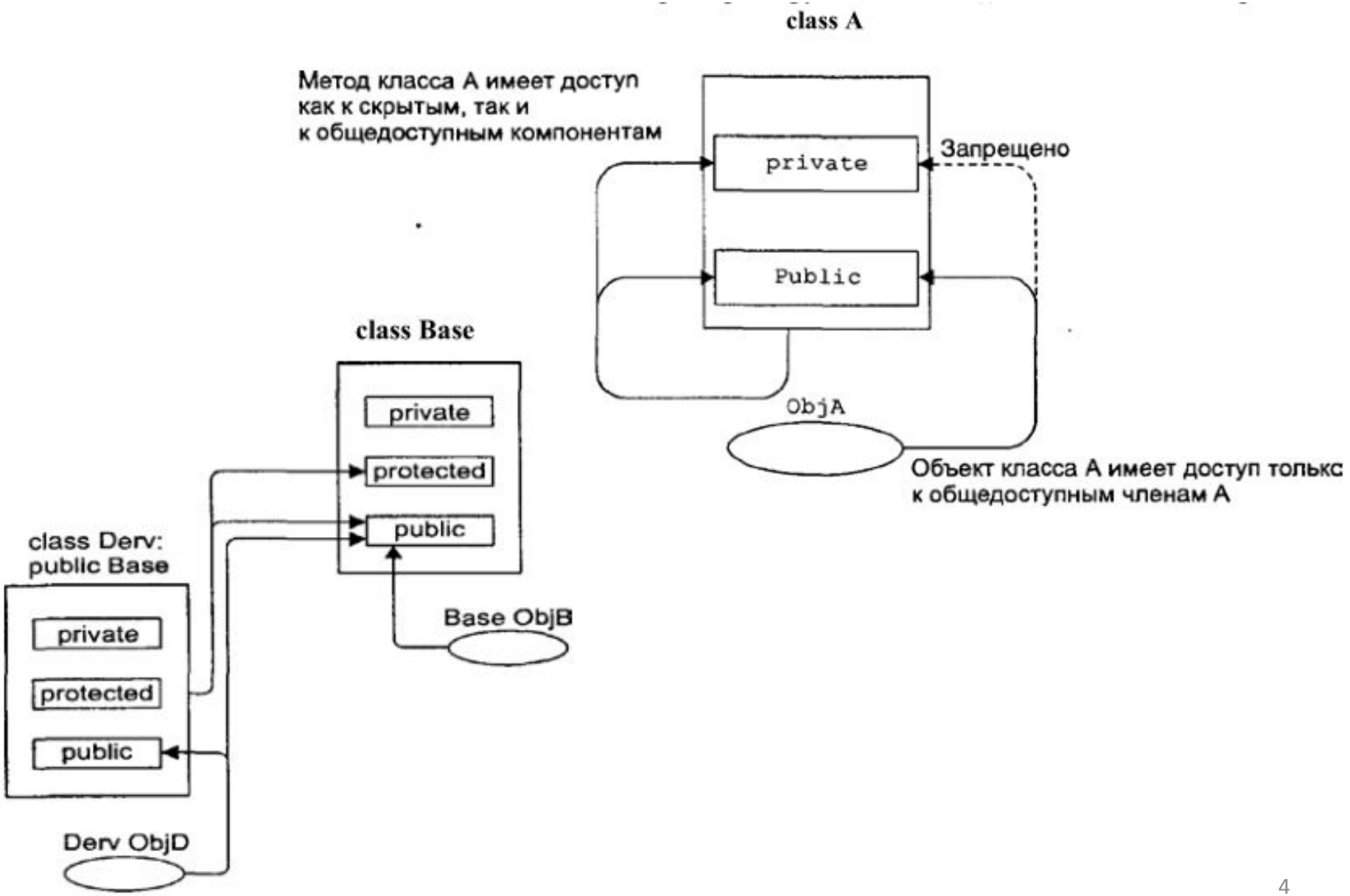
Наследование

Наследование

```
1  class Counter //базовый класс
2  {
3  protected:
4      unsigned int count; //счетчик
5  public:
6      Counter ( ) : count ( 0 ) //конструктор без аргументов
7      { }
8      Counter ( int c ) : count ( c )
9      { }
10     unsigned int get_count ( ) const
11     { return count; } // возвращает значение счетчика
12     Counter operator++ ( ) //увеличивает значение
13     //счетчика (префикс)
14     { return Counter ( ++count ); }
15 };
16 ////////////////////////////////////////////////////////////
17 class CountDn : public Counter//производный класс
18 {
19 public:
20     Counter operator-- ( ) //уменьшает значение счетчика
21     { return Counter ( --count ); }
22 };
23 ////////////////////////////////////////////////////////////
24 int main ( )
25 {
26     CountDn c1; // объект c1
27     cout << "\n c1=" << c1.get_count ( ); //вывод на печать
28     ++c1; ++c1; ++c1; //увеличиваем c1 три раза
29     cout << "\n c1=" << c1.get_count ( ); //вывод на печать
30     --c1; --c1; //уменьшаем c1 два раза
31     cout << "\n c1=" << c1.get_count ( ); //вывод на печать
32     cout << endl;
33     return 0;
34 }
```



Спецификатор доступа protected



Наследование и доступ

Специфиикатор доступа	Доступ из самого класса	Доступ из производных классов	Доступ из внешних классов и функций
public	Есть	Есть	Есть
protected	Есть	Есть	Нет
private	Есть	Нет	Нет

Конструкторы производного класса

```
1  class Counter
2  {
3  protected: // заметьте, что тут не следует использовать private
4      unsigned int count; // счетчик
5  public:
6      Counter ( ) : count ( ) // конструктор без параметров
7      { }
8      Counter ( int c ) : count ( c ) // конструктор с одним параметром
9      { }
10     unsigned int get_count ( ) const // получение значения
11     { return count; }
12     Counter operator++ ( ) // оператор увеличения
13     { return Counter ( ++count ); }
14 };
15 ////////////////////////////////////////////////////////////
16 class CountDn : public Counter
17 {
18 public:
19     CountDn ( ) : Counter ( ) // конструктор без параметров
20     { }
21     CountDn ( int c ) : Counter ( c )// конструктор с одним параметром
22     { }
23     CountDn operator-- ( ) // оператор уменьшения
24     { return CountDn ( --count ); }
25 };
26 ////////////////////////////////////////////////////////////
27 CountDn c1; // переменные класса CountDn
28 CountDn c2 ( 100 );
```

Конструктор с одним аргументом также используется в выражениях присваивания:

CountDn c3 = --c2;

Перегрузка функций

```
class Stack
{
protected: // Замечание: использовать private нельзя
    enum { MAX = 3 }; // размер стека
    int st[ MAX ]; // данные, хранящиеся в стеке
    int top; // индекс последнего элемента в стеке
public:
    Stack ( ) // конструктор
    { top = -1; }
    void push ( int var ) // помещение числа в стек
    { st[ ++top ] = var; }
    int pop ( ) // извлечение числа из стека
    { return st[ top-- ]; }
};
class Stack2 : public Stack
{
public:
    void push ( int var ) // помещение числа в стек
    {
        if( top >= MAX-1 ) // если стек полон, то ошибка
        { cout << "\nОшибка: стек полон"; exit ( 1 ); }
        Stack::push ( var ); // вызов функции push класса Stack
    }
    int pop ( ) // извлечение числа из стека
    {
        if ( top < 0 ) // если стек пуст, то ошибка
        { cout << "\nОшибка: стек пуст\n"; exit ( 1 ); }
        return Stack::pop ( ); // вызов функции pop класса Stack(можно без return)
    }
};
int main ( )
{
    Stack2 s1;
    s1.push ( 11 ); // поместим в стек несколько чисел
    s1.push ( 22 ); s1.push ( 33 );
    cout << endl << s1.pop ( ); // заберем числа из стека
    cout << endl << s1.pop ( ); cout << endl << s1.pop ( );
    cout << endl << s1.pop ( ); // ой, а данных-то больше нет
    return 0;
}
```

Перегрузка функций

```
enum posneg { pos, neg };
class Distance // класс для английских мер длины
{
protected: // заметьте, что private использовать нельзя
    int feet;
    float inches;
public:
    // конструктор без параметров
    Distance ( ) : feet ( 0 ), inches ( 0.0 )
    { }
    // конструктор с двумя параметрами
    Distance ( int ft, float in ) : feet ( ft ), inches ( in )
    { }
    // получение значений от пользователя
    void getdist ( )
    {
        cout << "\nВведите футы: "; cin >> feet;
        cout << "Введите дюймы: "; cin >> inches;
    }
    // вывод значений на экран
    void showdist ( ) const
    { cout << feet << "\'-" << inches << '\"'; }
};
```

```
class DistSign : public Distance // добавление знака к длине
{
private:
    posneg sign; // значение может быть pos или neg
public:
    // конструктор без параметров
    DistSign ( ) : Distance ( )
    { sign = pos; }
    // конструктор с двумя или тремя параметрами
    DistSign ( int ft, float in, posneg sg = pos ) :
    Distance ( ft, in ) // вызов конструктора базового класса
    { sign = sg; } // начальная установка знака
    void getdist ( ) // ввод пользователем длины
    {
        Distance::getdist ( ); // вызов функции getdist из ба
        char ch; // запрос знака
        cout << "Введите знак (+ или -): "; cin >> ch;
        sign = ( ch == '+' ) ? pos : neg;
    }
    void showdist ( ) const // вывод расстояния
    {
        // вывод всей информации, включая знак
        cout << ( ( sign == pos ) ? "(+)" : "(-)" );
        Distance::showdist ( );
    }
};
```

Иерархия классов

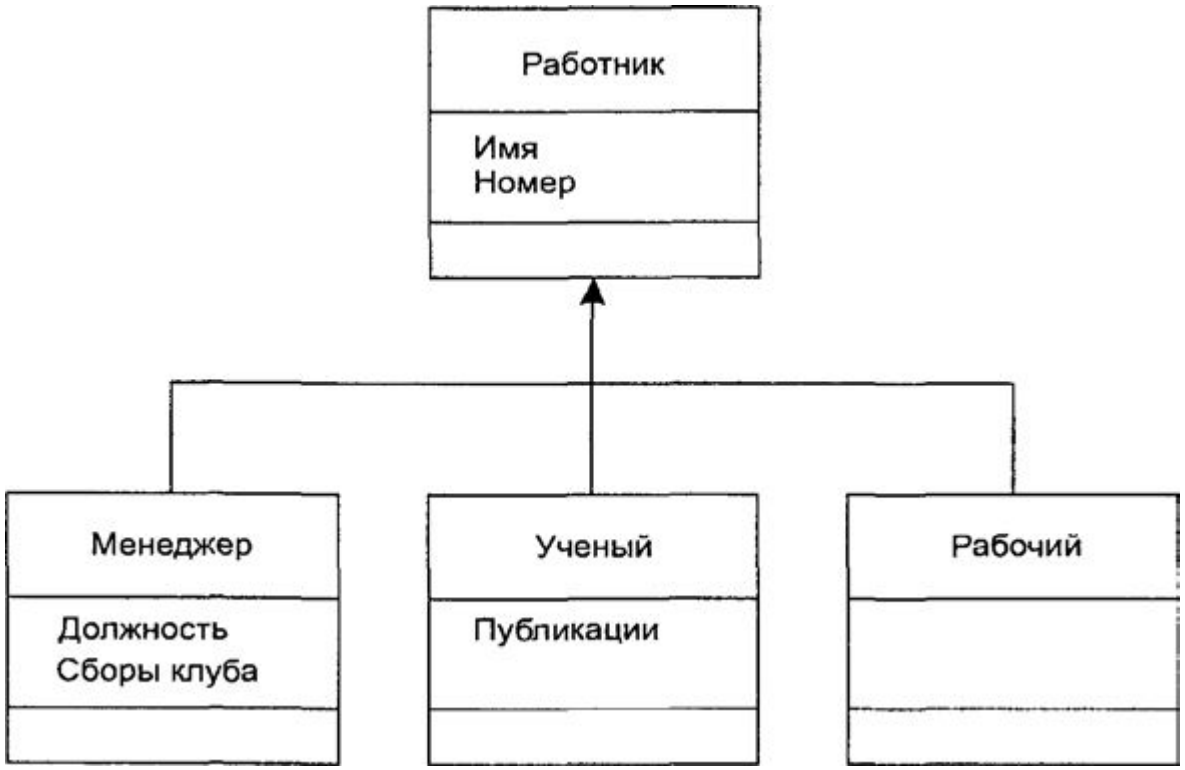


Диаграмма классов UML для примера EMPLOY

Иерархия классов

```
const int LEN = 80; // максимальная длина имени
class employee // некий сотрудник
{
private:
    char name[ LEN ]; // имя сотрудника
    unsigned long number; // номер сотрудника
public:
    void getdata ( )
    {
        cout << "\n Введите фамилию: "; cin >> name;
        cout << " Введите номер: "; cin >> number;
    }
    void putdata ( ) const
    {
        cout << "\n Фамилия: " << name;
        cout << "\n Номер: " << number;
    }
};
////////////////////////////////////////////////
class manager : public employee // менеджер
{
private:
    char title[ LEN ]; // должность, например
    double dues; // сумма взносов в гольф-клуб
public:
    void getdata ( )
    {
        employee::getdata ( );
        cout << " Введите должность: "; cin >> title;
        cout << " Введите сумму взносов в гольф-клуб: "; cin >> dues;
    }
    void putdata ( ) const
    {
        employee::putdata ( );
        cout << "\n Должность: " << title;
        cout << "\n Сумма взносов в гольф-клуб: " << dues;
    }
};
```

```
class scientist : public employee // ученый
{
private:
    int pubs; // количество публикаций
public:
    void getdata ( )
    {
        employee::getdata ( );
        cout << " Введите количество публикаций: "; cin >> pubs;
    }
    void putdata ( ) const
    {
        employee::putdata ( );
        cout << "\n Количество публикаций: " << pubs;
    }
};
////////////////////////////////////////////////////////
class laborer : public employee // рабочий
{
};
```

Иерархия классов

```
int main ( )
{
    manager m1, m2;
    scientist s1;
    laborer l1;
    // введем информацию о нескольких сотрудниках
    cout << endl;
    cout << "\nВвод информации о первом менеджере";
    m1.getdata ( );
    cout << "\nВвод информации о втором менеджере";
    m2.getdata ( );
    cout << "\nВвод информации о первом ученом";
    s1.getdata ( );
    cout << "\nВвод информации о первом рабочем";
    l1.getdata ( );
    // выведем полученную информацию на экран
    cout << "\nИнформация о первом менеджере";
    m1.putdata ( );
    cout << "\nИнформация о втором менеджере";
    m2.putdata ( );
    cout << "\nИнформация о первом ученом";
    s1.putdata ( );
    cout << "\nИнформация о первом рабочем";
    l1.putdata ( );
    cout << endl;
    return 0;
}
```

Иерархия классов

```
class shape // базовый класс
{
protected:
    int xCo, yCo; // координаты фигуры
    color fillcolor; // цвет
    fstyle fillstyle; // стиль изображения
public:
    // конструктор без аргументов
    shape ( ) : xCo ( 0 ), yCo ( 0 ), fillcolor ( cWHITE ), fillstyle ( SOLID_FILL )
    { }
    // конструктор с пятью аргументами
    shape ( int x, int y, color fc, fstyle fs ) : xCo ( x ), yCo ( y ),
        fillcolor ( fc ), fillstyle ( fs )
    { }
    // функция установки цвета и стиля
    void draw ( ) const
    {
        set_color ( fillcolor );
        set_fill_style ( fillstyle );
    }
};
////////////////////////////////////////////////////////
class circle : public shape
{
private:
    int radius; // радиус, а xCo и yCo будут координатами центра
public:
    // конструктор без параметров
    circle ( ) : shape ( )
    { }
    // конструктор с пятью параметрами
    circle ( int x, int y, int r, color fc, fstyle fs ) :
        shape ( x, y, fc, fs ), radius ( r )
    { }
    void draw ( ) const // функция рисования окружности
    {
        shape::draw ( );
        draw_circle ( xCo, yCo, radius );
    }
};
```

Иерархия классов

```
42  class rect : public shape
43  {
44  private:
45      int width, height; // ширина и высота, а xCo и yCo будут координатами верхнего пра
46  public:
47      // конструктор без параметров
48      rect ( ) : shape ( ), height ( 0 ), width ( 0 )
49      { }
50      // конструктор с шестью параметрами
51      rect ( int x, int y, int h, int w, color fc, fstyle fs ) : shape ( x, y, fc, fs ),
52          height ( h ), width ( w )
53      { }
54      void draw ( ) const// функция рисования прямоугольника
55      {
56          shape::draw ( );
57          draw_rectangle ( xCo, yCo, xCo + width, yCo + height );
58          // нарисуем диагональ
59          set_color ( xWHITE );
60          draw_line ( xCo, yCo, xCo + width, yCo + height );
61      }
62  };
63  ////////////////////////////////////////
64  class tria : public shape
65  {
66  private:
67      int height; // высота пирамиды, а xC
68  public:
69      // конструктор без параметров
70      tria ( ) : shape ( ), height ( 0 )
71      { }
72      // конструктор с пятью параметрами
73      tria ( int x, int y, int h, color fc, fstyle fs ) :
74          shape ( x, y, fc, fs ), height( h )
75      { }
76      // функция рисования пирамиды
77      void draw ( ) const
78      {
79          shape::draw ( );
80          draw_pyramid( xCo, yCo, height );
81      }
82  };
```

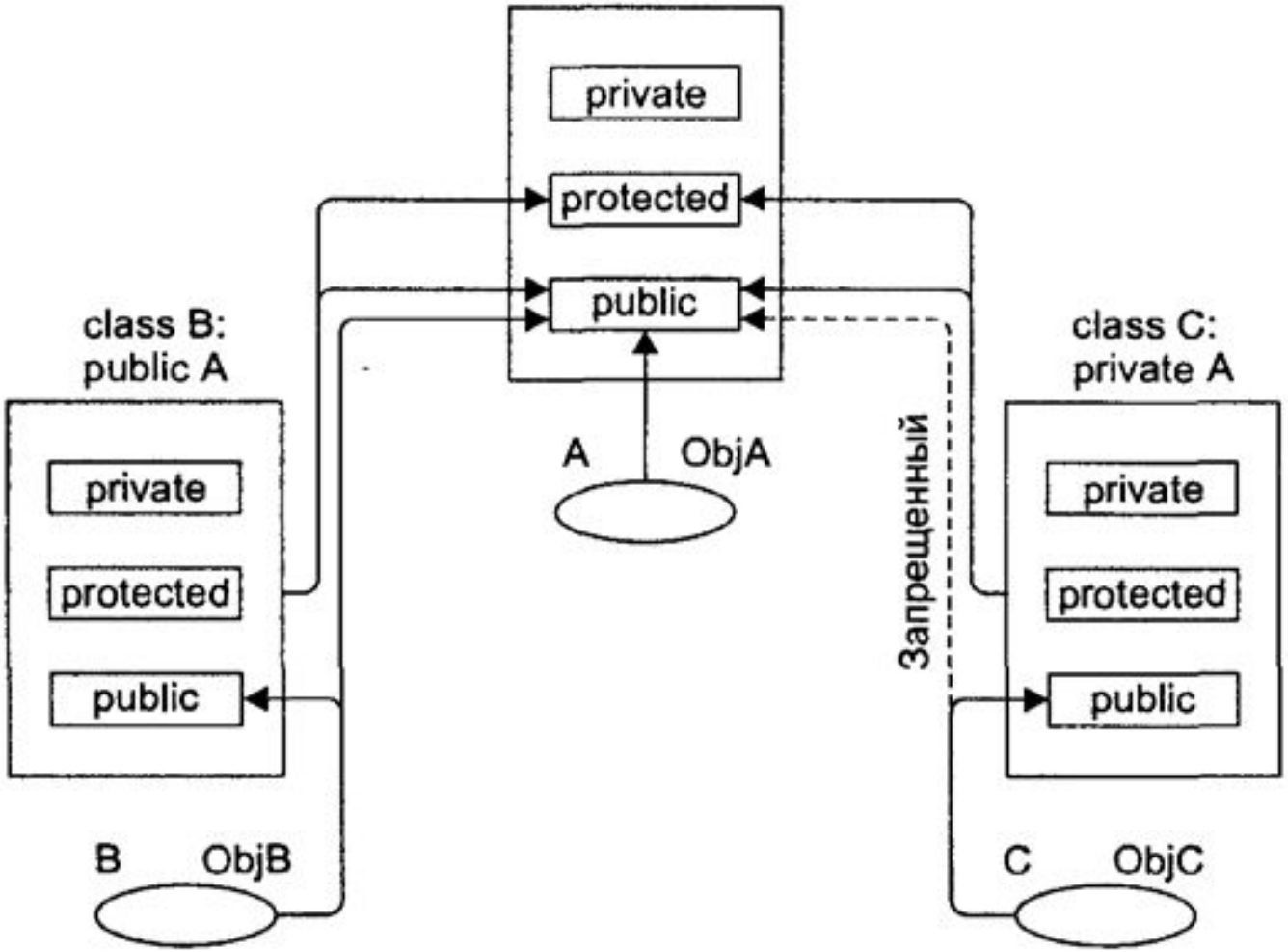
```
84  int main ( )
85  {
86      init_graphics ( ); // инициализируем систему отображения графики
87      circle cir ( 40, 12, 5, cBLUE, X_FILL ); // создаем круг
88      rect rec ( 12, 7, 10, 15, cRED, SOLID_FILL ); // создаем прямоугол
89      tria tri ( 60, 7, 11, cGREEN, MEDIUM_FILL ); // создаем пирамиду
90      // рисуем все наши фигуры
91      cir.draw ( );
92      rec.draw ( );
93      tri.draw ( );
94      set_cursor_pos( 1, 25 ); // переводим курсор в самый низ экрана
95      return 0;
96  }
```

Общее и частное наследование

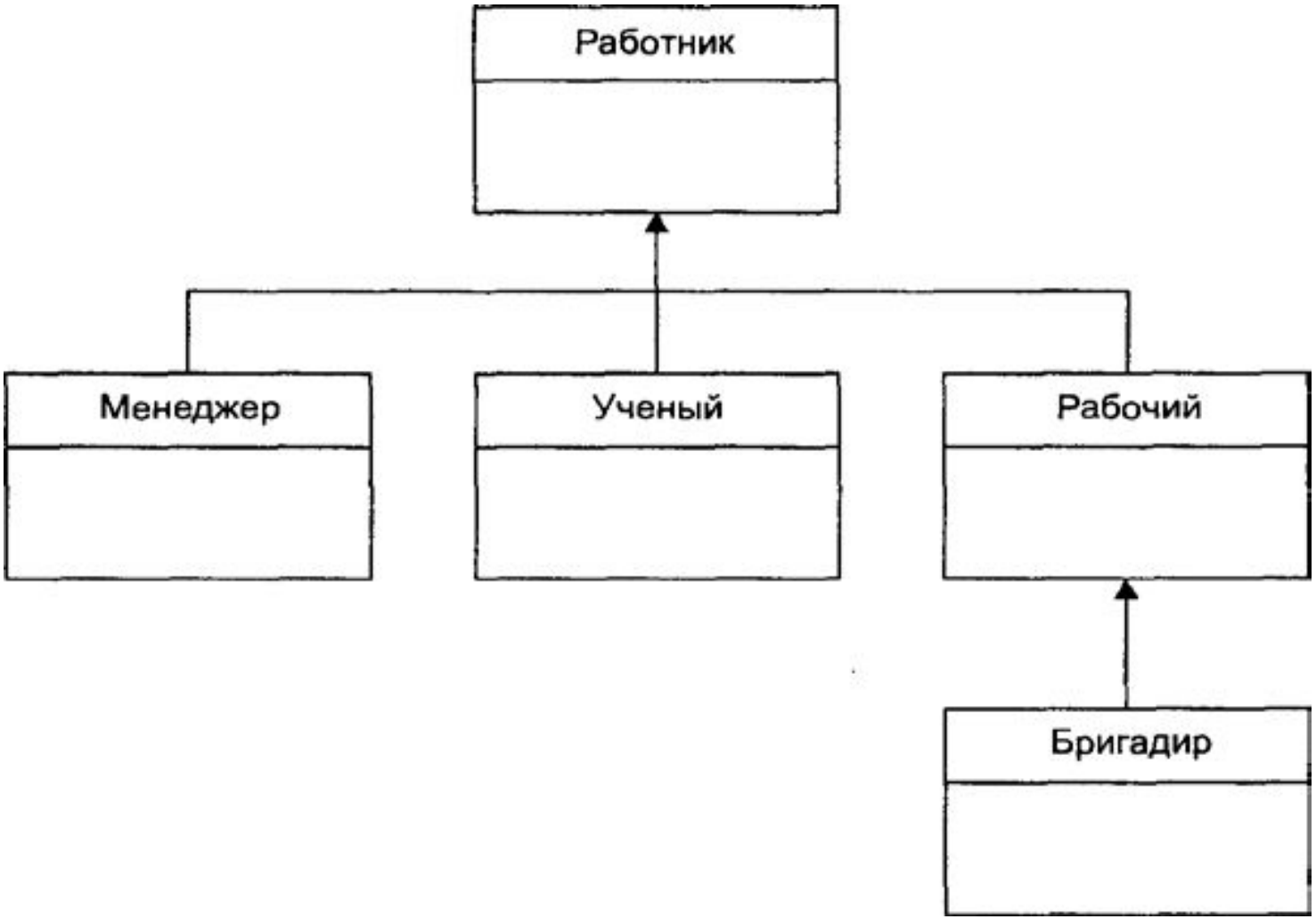
```
1  class A // базовый класс
2  {
3  private: // тип доступа к данным совпадает с типом
4      int privdataA; // доступа к функциям
5  protected:
6      int protdataA;
7  public:
8      int pubdataA;
9  };
10 ////////////////////////////////////////
11 class B : public A // public наследование
12 {
13 public:
14     void funct ( )
15     {
16         int a;
17         a = privdataA; // ошибка, нет доступа
18         a = protdataA; // так можно
19         a = pubdataA; // так можно
20     }
21 };
22 ////////////////////////////////////////////////
23 class C : private A // private наследование
24 {
25 public:
26     void funct ( )
27     {
28         int a;
29         a = privdataA; // ошибка, нет доступа
30         a = protdataA; // так можно
31         a = pubdataA; // так можно
32     }
33 };
```

```
35 int main ( )
36 {
37     int a;
38     B objB;
39     a = objB.privdataA; // ошибка, нет доступа
40     a = objB.protdataA; // ошибка, нет доступа
41     a = objB.pubdataA; // так можно
42     C objC;
43     a = objC.privdataA; // ошибка, нет доступа
44     a = objC.protdataA; // ошибка, нет доступа
45     a = objC.pubdataA; // ошибка, нет доступа
46     return 0;
47 }
```

Общее и частное наследование



Уровни наследования



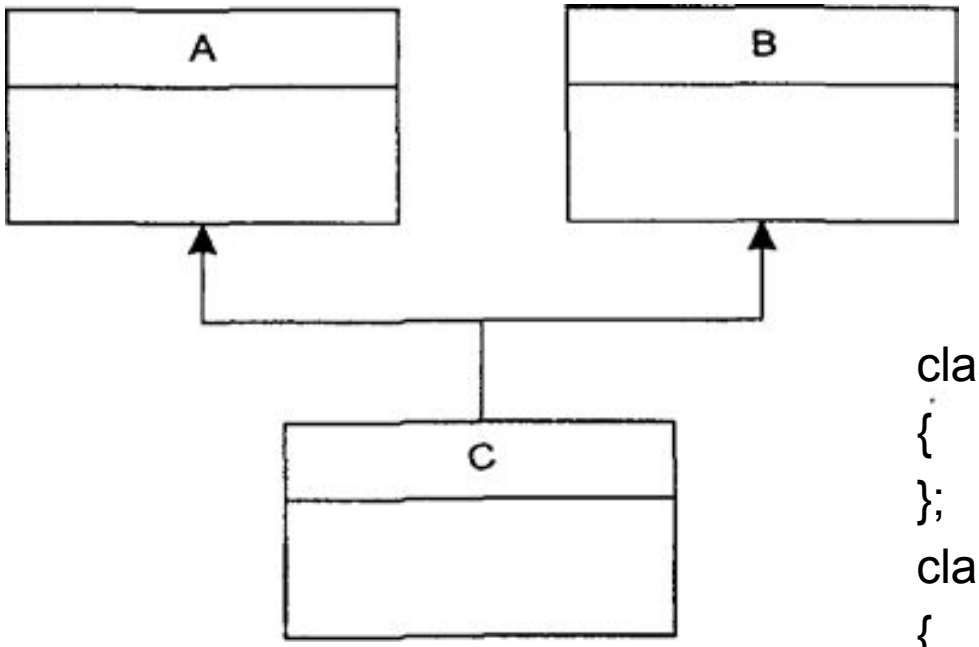
UML диаграмма классов программы EMPLOY2

Уровни наследования

```
const int LEN = 80;
//////////////////////////////////////////////////////////
class employee // некий сотрудник
{
private:
    char name[ LEN ]; // имя сотрудника
    unsigned long number; // номер сотрудника
public:
    void getdata ( );
    void putdata ( ) const;
};
//////////////////////////////////////////////////////////
class manager : public employee // менеджер
{
private:
    char title[ LEN ]; // должность, например вице-президент
    double dues; // сумма взносов в
public:
    void getdata ( );
    void putdata ( ) const;
};
//////////////////////////////////////////
class scientist : public employee /
{
private:
    int pubs; // количество публика
public:
    void getdata ( );
    void putdata ( ) const;
};
```

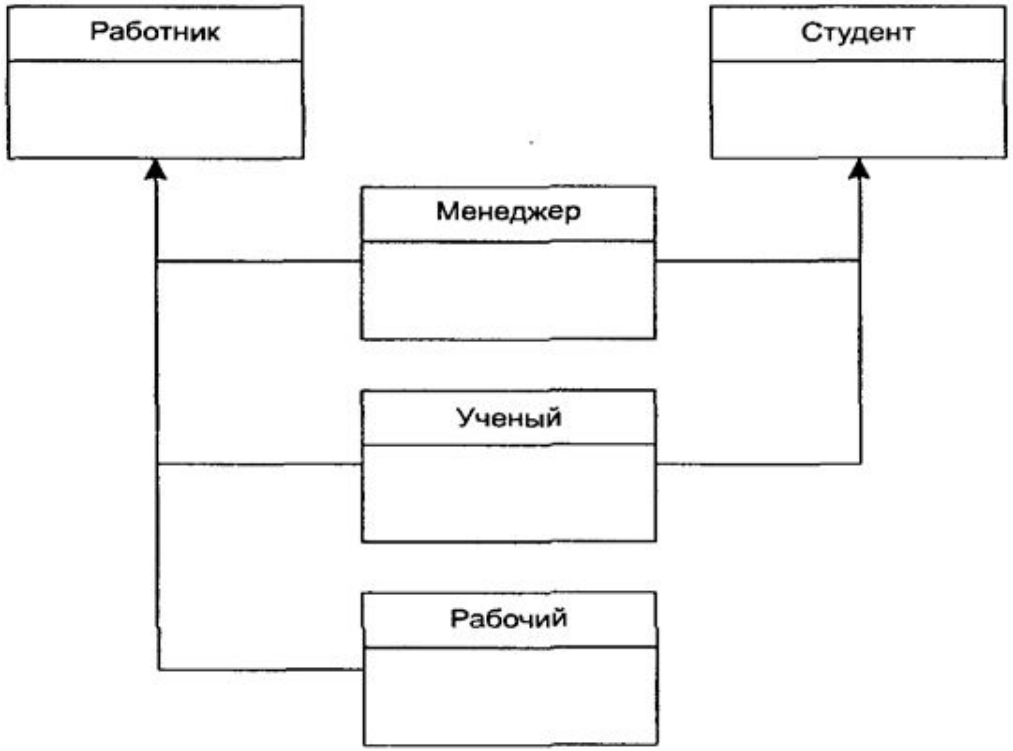
```
class laborer : public employee // рабочий
{
};
//////////////////////////////////////////////////////////
class foreman : public laborer // бригадир
{
private:
    float quotas; // норма выработки
public:
    void getdata ( )
    {
        laborer::getdata ( );
        cout << " Введите норму выработки: "; cin >> quotas;
    }
    void putdata ( ) const
    {
        laborer::putdata ( );
        cout << "\n Норматив: " << quotas;
    }
};
```

Множественное наследование




```
class A
{
};
class B
{
};
class C : public A, public B
{
};
```

Множественное наследование




```
1   class student
2   { };
3   class employee
4   { };
5   class manager : public employee, private student
6   { };
7   class scientist : private employee, private student
8   { };
9   class laborer : public employee
10  { };
```

Множественное наследование

```
class student // сведения об образовании
{
private:
    char school[ LEN ]; // название учебного заведения
    char degree[ LEN ]; // уровень образования
public:
    void getedu ( )
    {
        cout << " Введите название учебного заведения: ";
        cin >> school;
        cout << " Введите степень высшего образования\n";
        cout << " (неполное высшее, бакалавр, магистр, кандидат наук): ";
        cin >> degree;
    }
    void putedu ( ) const
    {
        cout << "\n Учебное заведение: " << school;
        cout << "\n Степень: " << degree;
    }
};

class manager : private employee, private student // менеджер
{
private:
    char title[ LEN ]; // должность сотрудника
    double dues; // сумма взносов в гольф-клуб
public:
    void getdata ( )
    {
        employee::getdata ( );
        cout << " Введите должность: "; cin >> title;
        cout << " Введите сумму взносов в гольф-клуб: "; cin >> dues;
        student::getedu ( );
    }
    void putdata ( ) const
    {
        employee::putdata ( );
        cout << "\n Должность: " << title;
        cout << "\n Сумма взносов в гольф-клуб: " << dues;
        student::putedu ( );
    }
};
```

Конструкторы при множественном наследовании

```
class Type //Тип древесины
{
private:
    string dimensions;
    string grade;
public: //конструктор без параметров
    Type() : dimensions("N/A"), grade("N/A")
    { }
    //конструктор с двумя параметрами
    Type(string di, string gr) : dimensions(di), grade(g
    { }
    void gettype() //запрос информации у пользователя
    {
        cout << " Введите номинальные размеры(2x4 и т.д.
        cin >> dimensions;
        cout << " Введите сорт(необработанная,брус и т.д
        cin >> grade;
    }
    void showtype() const //показ информации
    {
        cout << "\n Размеры: " << dimensions;
        cout << "\n Сорт: " << grade;
    }
};
//////////////////////////////////////////////////////////
class Distance //английские меры длины
{
private:
    int feet;
    float inches;
public: //конструктор без параметров
    Distance() : feet(0), inches(0.0)
    { } //конструктор с двумя параметрами
    Distance(int ft, float in) : feet(ft), inches(in)
     { }
    void getdist() //запрос информации у пользователя
    {
        cout << " Введите футы: "; cin >> feet;
        cout << " Введите дюймы: "; cin >> inches;
    }
    void showdist() const //показ информации
    { cout << feet << "\'-" << inches << '\"'; }
};
```

```
class Lumber : public Type, public Distance
{
private:
    int quantity; //количество штук
    double price; //цена за штуку
public: //конструктор без параметров
    Lumber() : Type(), Distance(), quantity(0), price(0.0)
    { }
    //конструктор с шестью параметрами
    Lumber( string di, string gr, //параметры для Type
        int ft, float in, //параметры для Distance
        int qu, float prc ) : //наши собственные параметры
        Type(di, gr), //вызов конструктора Type
        Distance(ft, in), //вызов конструктора Distance
        quantity(qu), price(prc)//вызов наших конструкторов
    { }
    void getlumber()
    {
        Type::gettype();
        Distance::getdist();
        cout << " Введите количество: "; cin >> quantity;
        cout << " Введите цену: "; cin >> price;
    }
    void showlumber() const
    {
        Type::showtype();
        cout << "\n Длина: ";
        Distance::showdist();
        cout << "\n Стоимость " << quantity
        << " штук: $" << ( price * quantity ) << " рублей";
    }
};
```

```
int main()
{
    Lumber siding; //используем конструктор без параметров
    cout << "\n Информация об обшивке:\n";
    siding.getlumber(); //получаем данные от пользователя
    //используем конструктор с шестью параметрами
    Lumber studs( "2x4", "const", 8, 0.0, 200, 4.45F );
    //показываем информацию
    cout << "\nОбшивка"; siding.showlumber();
    cout << "\nБрус"; studs.showlumber();
    cout << endl;
    return 0;
}
```

Неопределенность при множественном наследовании

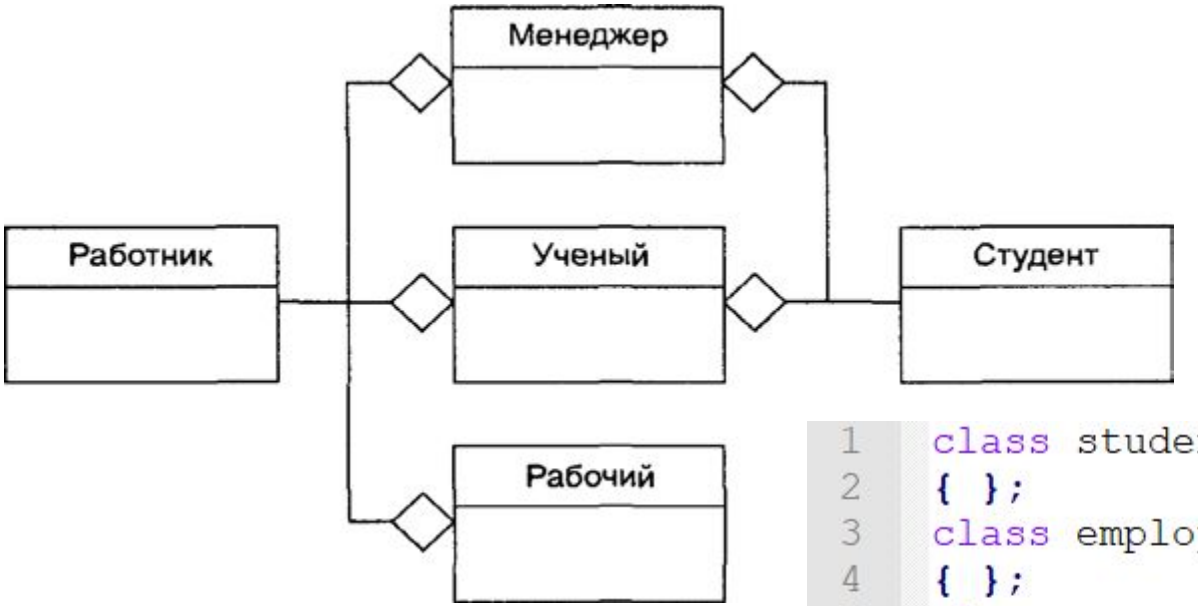
```
class A
{
public:
    void show ( ) { cout << "Класс A\n"; }
};
class B
{
public:
    void show ( ) { cout << "Класс B\n"; }
};
class C : public A, public B
{
};
//////////////////////////////////////////////////////////
int main ( )
{
    C objC; // объект класса C
    // objC.show ( ); // так делать нельзя - программа не скомпилируется
    objC.A::show ( ); // так можно
    objC.B::show ( ); // так можно
    return 0;
}
```

```
class A
{
public:
    virtual 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;
}
```

Включение: классы в классах

```
class A
{ };
class B
{
      A objA;
};
```

Включение: классы в классах




```
1   class student
2   { };
3   class employee
4   { };
5   class manager
6   {
7       student stu;
8       employee emp;
9   };
10  class scientist
11  {
12      student stu;
13      employee emp;
14  };
15  class laborer
16  {
17      employee emp;
18  }
```

Композиция: сложное включение

Композиция — это более сложная форма объединения. Она обладает всеми его

свойствами, но имеет еще и такие, как:

- часть может принадлежать только одному целому;
- время жизни части то же, что и целого.