Семинар 11
Наследование в Си++

Отношения включения (composition) и наследования (inheritance)

Нужно различать отношения включения и наследования классов.
Включение = has a = «имеет как часть»
Наследование = is a = «является частным случаем»

Автомобиль имеет (has a) мотор
Автомобиль является (is a) транспортным средством

Отношения включения и наследования

В связи с различиями в отношениях «is a» и «has a» крайне нежелательно наследовать, например, класс «автомобиль» от класса «двигатель», т.к. автомобиль не является частным случаям двигателя, а включает его как часть.

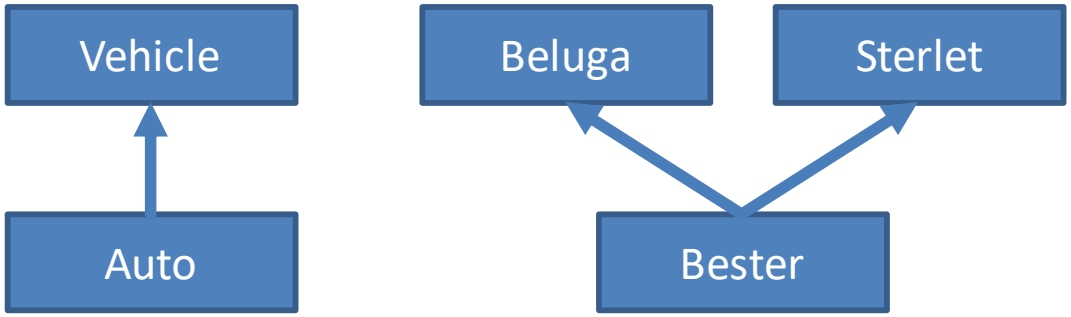
Отношения включения

Пример отношения включения классов:

```
class point{
        double x;
        double y;
        public:
        point(double x0, double y0) :x(x0), y(y0) {}
};
// Класс circle включает поле center класса point
class circle{
        point center;
        double radius;
        public:
        circle(point c, double r) :center(c), radius(r) {} // Вызов констр. копир.
        circle(double x0, double y0, double r) :center(x0, y0), radius(r) {}
};
```

Отношения наследования

При наследовании выстраиваются отношения как минимум между двумя классами. Первый класс – *базовый*, второй класс – *производный*. В случае, если базовых классов несколько, то наследование называется *множественным*.



Пример множественного наследования: бестер есть гибрид белуги и стерляди

Отношения наследования

Класс называется *производным или наследником* базового класса, если он *описывает специфическое подмножество* тех объектов, которые определяются базовым классом. В связи с этой идеей появился принцип подстановки Барбары Лисков: «Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом»

Отношения наследования

Принцип подстановки при программировании в Си++ выполняется не всегда: в Си++ можно не только конкретизировать более общее понятие, дополняя его специфическими полями и методами, но и сужать базовое понятие, исключая некоторые из его свойств.

Отношения наследования

При наследовании производный класс получает поля данных и методы базового класса и может быть дополнен новыми полями данных и методами.

В простейшем случае спецификация производного класса имеет вид:

class имя_произв_класса: public имя_базового_класса
{поля данных и методы производного класса};

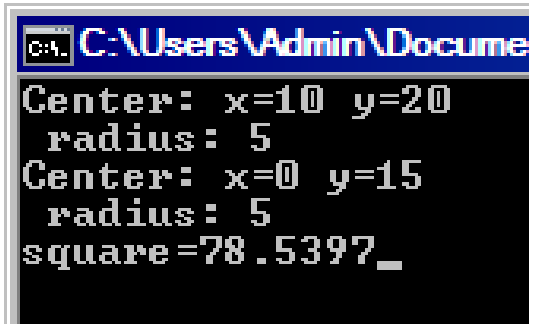
Отношения наследования

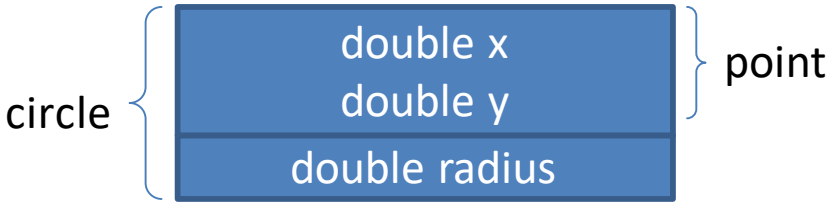
```
class point{
      double x; // Эти поля будут недоступны в производном классе
      double y;
public:
      point(double x0, double y0) :x(x0), y(y0) {}
      void move(double dx, double dy){x+=dx; y+=dy;}
      void display() {cout<<"x="<<x<<" y="<<y<<endl;}
};

class circle : public point{
      double radius;
public:
      circle(double x0, double y0, double r) :point(x0, y0), radius(r) {}
      void display()
      {cout<<"Center:\t"; point::display(); cout<<" radius: "<<radius<<endl;}
      double square() {return 3.14159*radius*radius;}
};
```

Отношения наследования

```
int _tmain(int argc, _TCHAR* argv[])
{
	circle c1(10, 20, 5.0);
	c1.display(); //Переопределённый метод класса ellipse
	c1.move(-10.0, -5.0); //Метод класса point
	c1.display();
	cout<<"square="<<c1.square();
	getchar();
	return 0;
}
```





Отношения наследования

При наследовании:

• производный класс беспрепятственно обращается к доступным для него полям данных и методам базового класса

• базовый класс не имеет доступа к полям данных и методам производного класса

• в объект производного класса включаются поля данных базового класса, т.е. можно считать, что в объект производного класса входит экземпляр объекта базового класса

• если в производном классе имя поля данных или метода базового класса использовано для других целей, то соответствующий компонент базового класса доступен в производном классе с помощью указания области видимости: имя_базового_класса::имя_компонента (см. point::display();)

• наследуемая функция базового класса в производном классе может: 1) остаться без изменений, 2) быть заменена (как в случае с методом display), 3) в производном классе можно перегрузить функцию (определить с тем же именем, но с другой спецификацией параметров)

Доступность компонентов при наследовании

Сокращённый формат спецификации производного класса:
ключ_класса имя_класса : список_спецификаторов_баз
{поля данных и методы производного класса};

ключ_класса – struct или class (не union!)

Спецификаторы базы разделяются запятыми и могут быть:

1. спецификатор_доступа имя_класса
2. virtual спецификатор_доступа имя_класса
3. спецификатор_доступа virtual имя_класса

спецификатор_доступа – одно из ключевых слов public, protected, private.
Спецификатор доступа не является обязательным

Доступность компонентов при наследовании

При наследовании доступность поля/метода базового класса в производном классе определяют:

1. доступность поля/метода в базовом классе
2. спецификатор доступа базового класса (в строке спецификации производного)
3. ключ производного класса

Вспомним:

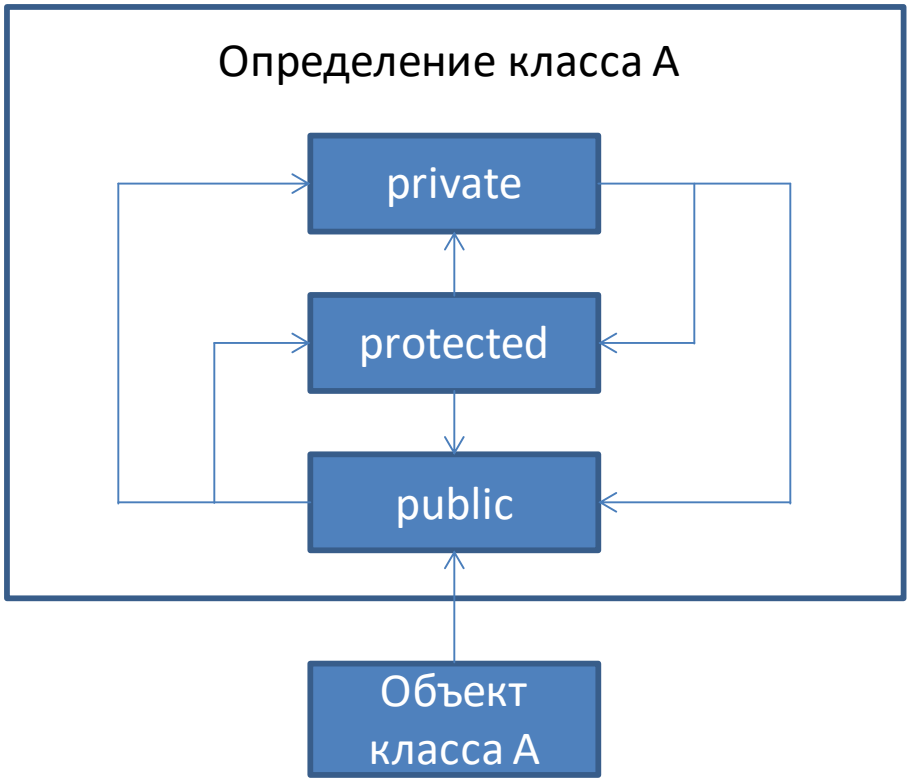
private означает, что компоненты закрыты и к ним можно обратиться только из определения класса и дружественных фукций/классов

publuc означает общедоступность/открытость компонентов, к ним можно обращаться как внутри определения класса, так и вне его через объекты класса

protected означает, что компонент объявлен как защищённый, т.е. к данному компоненту можно обратиться как из данного класса, так и из его потомков, но не через объект класса.

Доступность компонентов при наследовании

Рис. Доступность компонентов для объектов и методов класса (без наследования).



Главная идея:

- внутри класса все поля/методы видимы для всех методов
- за пределами класса объект (экземпляр) класса может обращаться только к открытым (private) полям и методам

Доступность компонентов при наследовании

Таблица доступности унаследованных компонентов

Доступность в базовом классе	Спецификатор доступа базового класса	struct (ключ класса-наследника)		class (ключ класса-наследника)	
public protected private	отсутствует	public protected недоступны	открыт	private private недоступны	закр
public protected private	public	public protected недоступны	открыт	public protected недоступны	открыт
public protected private	protected	protected protected недоступны	защ	protected protected недоступны	защ
public protected private	private	private private недоступны	закр	private private недоступны	закр

Доступность компонентов при наследовании

Примеры к таблице:

class B {private: long l; protected: int i; public: char c;}

class E : B {…}; // i, c – private (закрытое наследование)

struct S : B {…}; // c – public, i – protected (открытое наследование)

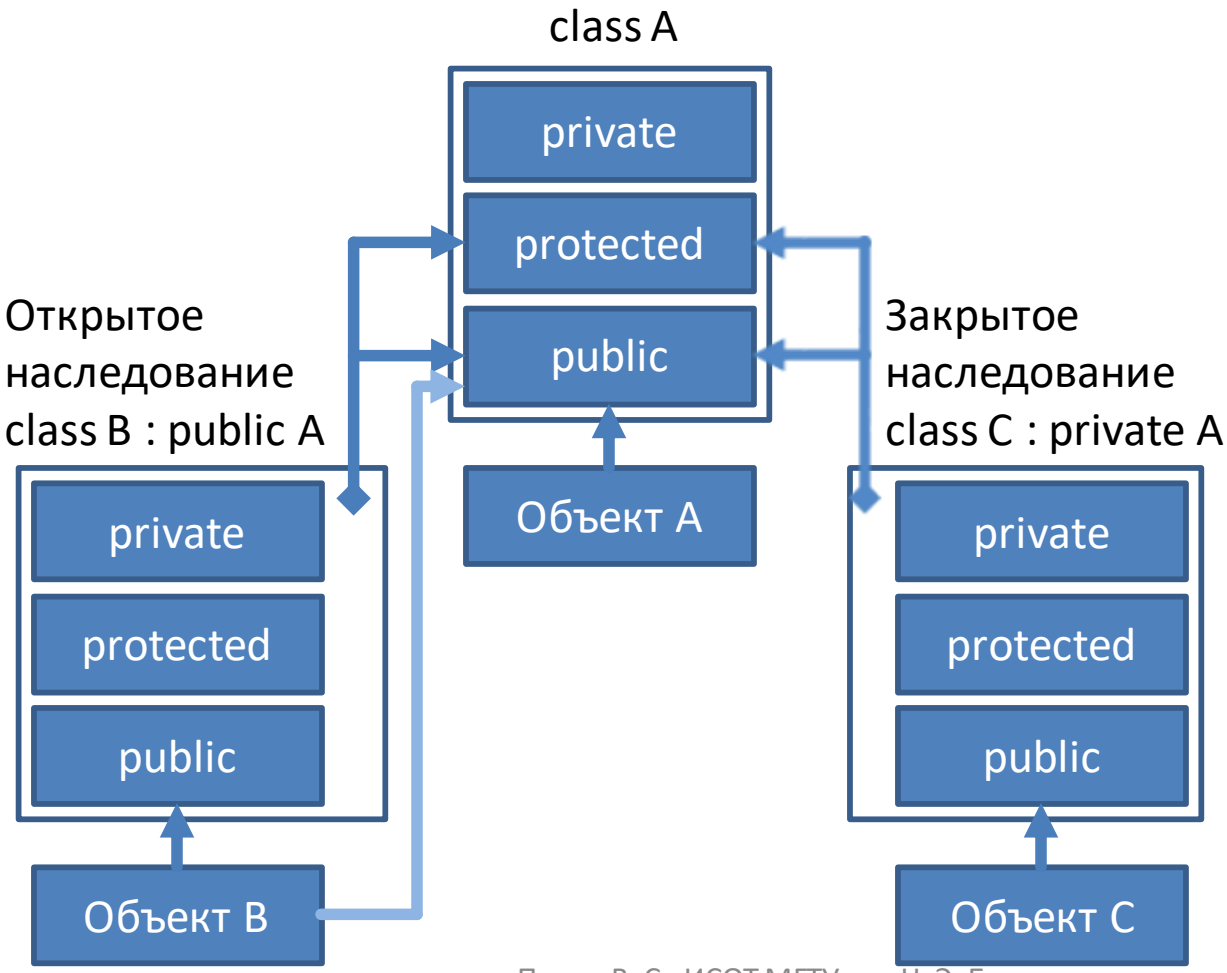
class F : protected B {…}; // i, c – protected (защищённое наследование)

class P : public B {…}; // c – public, i – protected (открытое наследование)

class Q : private B {…}; // i, c – private (закрытое наследование)

Доступность компонентов при наследовании

Рис. Разница между открытым и закрытым наследованием.



Доступность компонентов при наследовании

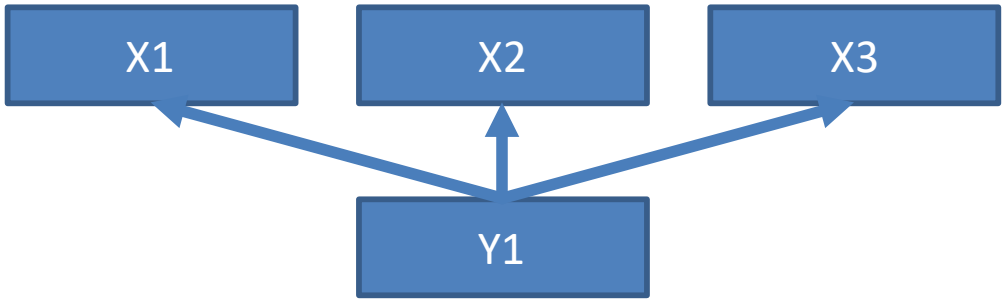
При закрытом или защищённом наследовании любые открытые методы и поля данных базового класса можно выборочно сделать открытыми и в производном классе. Для этого в производном классе нужно указать:

public: using имя_базового_класса : имя_компонента;

Множественное наследование и виртуальные классы

Наличие нескольких прямых базовых классов называют множественным наследованием:

```
class X1{…};
class X2{…};
class X3{…};
class Y1: public X1, public X2, public X3 {…};
```

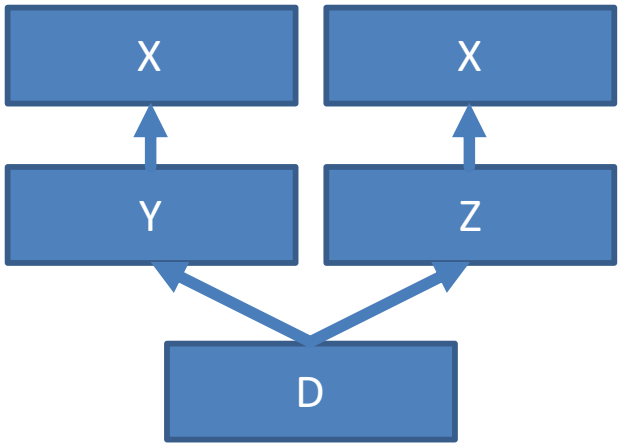


При множественном наследовании никакой класс не может больше одного раза использоваться в качестве прямого базового (т.е. «класса-родителя», входящего в список базовых при определении класса). Однако класс может больше одного раза быть непрямым базовым классом.

Множественное наследование и виртуальные классы

Пример дублирования непрямого базового класса:

```
class X {long double ax;};
class Y : public X {double ay;};
class Z : public X {int az;};
class D : public Y, public Z {};
```



Поле данных ax класса X дважды входит в объект класса X.

sizeof(X) = 12 (long double)

sizeof(Y) = 20 (long double + double)

sizeof(Z) = 16 (long double + int)

sizeof(D) = 36 (long double + long double + double + int)

Для обращения к полям данных ax нужно указывать полные квалифицированные имена:

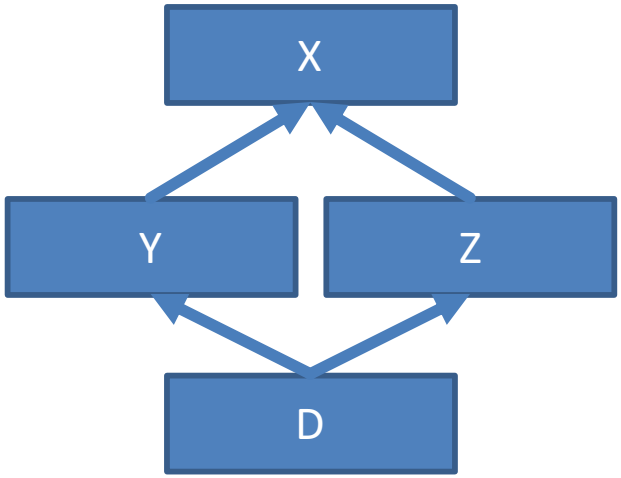
D :: Y :: X :: ax или

D :: Z :: X :: ax

Множественное наследование и виртуальные классы

- Как избежать дублирования непрямого базового класса?
- Указывать, что класс наследуется как виртуальный, используя ключевое слово virtual:

```
class X {long double ax;};
class Y : virtual public X {double ay;};
class Z : virtual public X {int az;};
class D : public Y, public Z {};
```



Для реализации виртуального наследования в производный класс компилятор включает в качестве поля данных указатель на виртуальный базовый класс.

sizeof(X) = 12 (long double)
sizeof(Y) = 24 (long double + double + x*)
sizeof(Z) = 20 (long double + int + x*)
sizeof(D) = 32 (long double + double + int + x* + x*)

Ещё раз о принципе подстановки

В соответствии с принципом подстановки каждый объект производного класса может быть использован в любой ситуации место объекта базового класса. Создадим два класса – окружность и круг. В приведённых примерах метод move базового класса может быть использован для производного без изменения, но метод compress необходимо экранировать в производном классе, т.к. при уменьшении/увеличении радиуса изменяется свойство производного класса.

Ещё раз о принципе подстановки

Реализация классов «окружность» и «круг»

```
class circle{
protected:
  double rad, len;
  int xc, yc;
public:
  circle(double r=0.0, int xcc=0, int ycc=0)
    :rad(r), xc(xcc), yc(ycc), len(2*3.14159*r) {}
  void move(int dx, int dy)
    {xc+=dx; yc+=dy;}
  circle compress(double k)
    {rad*=k; len*=k; return *this;}
};
```

```
class disk : public circle{
  double sqr;
public:
  disk(double r=0, int xcc=0, int ycc=0)
    :circle(r, xcc, ycc), sqr(3.14159*r*r) {}
  disk(const circle & c)
    :circle(c) {sqr = 3.14159*rad*rad;}
  disk compress(double k){
    circle::compress(k);
    sqr*=k*k;
    return *this;
  }
};
```

Задания

Задания:

1. Создать класс point (точка) и производные от него классы rectangle (прямоугольник) и ellipse (овал) двумя путями: пусть класс ellipse будет создан с помощью инструмента наследования, а класс rectangle – с помощью включения класса point.
2. Создать классы square (квадрат) и circle (окружность) как дочерние классы rectangle и ellipse соответственно.