

# ООП

## ООП в C++.

### Наследование. Полиморфизм

### Черновик

Кафедра ИВТ и ПМ

2018

# План

Прошлые темы

Наследование (Inheritance)

Простое наследование

Перегрузка и перекрытие методов

Динамический полиморфизм

Виртуальные методы

Ссылки и литература

# Outline

## Прошлые темы

### Наследование (Inheritance)

- Простое наследование

- Перегрузка и перекрытие методов

### Динамический полиморфизм

- Виртуальные методы

### Ссылки и литература

## Прошлые темы

- ▶ Опишите парадигму ООП
- ▶ Чем она отличается от парадигмы процедурного и модульного программирования?
- ▶ Из каких элементов строится программа написанная согласно парадигме объектно-ориентированного программирования ?
- ▶ Что такое класс?
- ▶ Что такое объект?
- ▶ Чем отличается класс от объекта?
- ▶ Что такое поле класса?
- ▶ Что такое метод класса?

## Прошлые темы

- ▶ Для чего нужен `this`?
- ▶ Какие модификаторы доступа могут применяться к атрибутам класса?

# Outline

Прошлые темы

## Наследование (Inheritance)

Простое наследование

Перегрузка и перекрытие методов

Динамический полиморфизм

Виртуальные методы

Ссылки и литература

# Outline

Прошлые темы

Наследование (Inheritance)

Простое наследование

Перегрузка и перекрытие методов

Динамический полиморфизм

Виртуальные методы

Ссылки и литература

# Наследование (Inheritance)

**Наследование** - построение новых классов на основе уже существующих.

**Базовый класс (предок)** — класс на основе которого строится определение нового класса - **производного класса (потомка)**.



# Наследование? Зачем?

Опишем класс для вектора  $V = (V_x, V_y)$  на плоскости

```
// класс - вектор на плоскости
class Vector2D{
    float _x, _y;
public:
    Vector2D();
    void setX(float x) {_x = x;}
    void setY(float y) {_y = y;};
    float x() const {return _x;}
    float y() const {return _y;}
    float abs() const {return sqrt(_x*_x + _y*_y);}
};
```

Что если потребуется создать класс для представления вектора  $V = (V_x, V_y, V_z)$ ? Придётся писать часть кода заново?

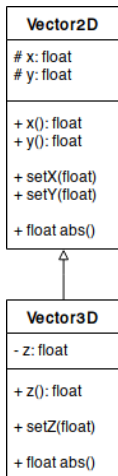
# Наследование? Зачем?

Если потребуется создать класс для представления вектора  $V = (V_x, V_y, V_z)$  то новый класс можно построить на основе старого, в котором уже будут методы и поля старого класса.

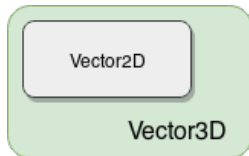
Наследование позволяет построить новый класс на основе имеющегося, *добавив* в него новые поля и методы.

# Наследование

## UML диаграмма классов



Любой экземпляр класса **Vector3D** теперь включает в себя всё, что есть в классе **Point2D**



В диаграмме выше для класса **Vector3D** приведены только его собственные поля, без полей унаследованных от **Vector2D**

# Наследование

```
class Vector2D{ // вектор на плоскости
protected:
    float _x, _y; // компоненты вектора
public:
    Vector2D() {}
    void setX(float x) {_x = x;}
    void setY(float y) {_y = y;}
    float x() const {return _x;}
    float y() const {return _y;}
    float abs() const {return sqrt(_x*_x + _y*_y);}
};

class Vector3D : public Vector2D{ // построим новый класс на основе Vector2D
    float _z; // поля _x и _y унаследованы
public: // x(), setX() и др. методы тоже унаследованы
    Vector3D() {}
    void setZ(float z) {_z = z;}
    float z() const {return _z;}
    // метод вычисления длины вектора здесь должен быть свой
    float abs() {return sqrt(_x*_x + _y*_y + _z*_z);}
};

Vector3D v1;
v1.setX(3);
v1.setZ(4);
v1.abs(); // 5
```

# Преобразование типов при наследовании

```
Vector2D v1(10, 20);  
Vector2D *v11;
```

```
Vector3D v2(100, 200, 300);  
Vector3D *v22;
```

*// так можно. но все, что не входит в Vector2D будет отброшено*

```
v1 = v2;
```

```
v11 = &v2; // и так можно.
```

```
v11->setX(42); // v2 = (42, 200, 300). но z так не поменять
```

*// а так нельзя: откуда взять z?*

```
v2 = v1;
```

*// это тоже нельзя*

```
v22 = v11;
```

## Наследование. Пример 2

```
class B{
    int x_;
protected:
    int y;
public:
    B() { cout << "Base constructor";
        x_ = 42; y = 9000;}
    void setX(int x_) {x = x_;}
    int x() const {return x_;}
    int getY() {return y;}
    void foo() const {cout << "Base";}
};
```

```
class D : public B{
    // поле x_ унаследовано,
    // но к нему нет прямого доступа
    // к полю y есть прямой доступ
    // только внутри этого класса
public:
    // в списке инициализации возможен
    // вызов конструктора базового класса
    D() : B() { setX(1729); }
    void bar() const
        {cout << "Delivered";}
};
```

```
B base;
D del;
base.foo();    // Base
// вызов унаследованного метода
del.foo();     // Base
del.bar();     // Delivered
del.x();       // 1729
del.getY();    // 9000
```

# Наследование и конструкторы, деструкторы ...

Эти методы хоть и наследуются, но не избавляют от написания аналогичных в производном классе:

- ▶ Конструкторы
- ▶ Деструктор
- ▶ Операторы присваивания

Например в конструкторе производного класса можно вызывать конструктор базового класса, но нельзя вызывать второй *вместо* первого.

# Наследование и конструкторы, деструкторы ...

- ▶ Конструктор по умолчанию базового класса вызывается автоматически перед вызовом конструктора производного класса.
- ▶ Если базовых классов несколько (многоуровневое наследование) то сначала вызывается конструктор самого базового класса.
- ▶ Деструкторы вызываются в обратном порядке: от производного класса к базовым



# Наследование и конструкторы, деструкторы ...

```
class Vector2D{  
    public:  
        Vector2D(float x, float y) {_x = x; _y = y;}  
        // ...  
};
```

```
class Vector3D : public Vector2D{  
    public:  
        // Вызов конструктора базового класса в конструкторе производного  
        Vector3D(float x, float y, float z) : Vector2D(x,y) {_z = z;}  
        // ...  
};
```

# Наследование и операторы

```
class Vector2D{  
    public:  
    // ...  
    Vector2D operator + (const Vector2D& v);  
};
```

```
class Vector3D : public Vector2D{  
    // ...  
};
```

```
Vector3D v1, v2;  
Vector3D v3 = v1 + v2;
```

Компилируется?

# Наследование и операторы

```
class Vector2D{  
    public:  
    // ...  
    Vector2D operator + (const Vector2D& v);  
};
```

```
class Vector3D : public Vector2D{  
    // ...  
};
```

```
Vector3D v1, v2;  
Vector3D v3 = v1 + v2;
```

Компилируется?

Ошибка: не определено оператора сложения для класса Vector3D.

Операторы наследуются. Однако в примере выше нужен оператор для класса Vector3D, однако унаследованный оператор принимает Vector2D

Ошибка станет очевиднее, если записать вызов оператора как `Vector3D v3 = v1.operator + (v2);`

# Наследование и модификаторы наследования

When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

# Outline

Прошлые темы

Наследование (Inheritance)

Простое наследование

Перегрузка и перекрытие методов

Динамический полиморфизм

Виртуальные методы

Ссылки и литература

# Перекрытие (overriding) и перегрузка (overloading) имён

Перекрытие имён в классе = переопределение имен

```
class B {  
    public:    void foo() {cout << "base";}   
              void bar() {cout << "bar";}   
              void bar(string s) {cout << "bar " << s;}      };  
  
class D: public B{  
    public: void foo(){cout << "delivered";}   
           void bar(int x) {cout << "bar";}      };
```

```
B base;  
D delivered;  
base.foo();           // base  
delivered.foo();      // delivered  
// Если нужно вызывать метод базового класса в производном:  
delivered.B::foo();   // base  
base.bar();           // bar  
base.bar("x");         // barx  
delivered.bar();      // bar  
delivered.bar(42);     // bar42
```

# Множественное наследование

Множественное наследование - наследование от нескольких классов одновременно.

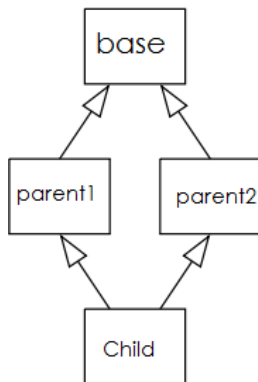
```
class Z: public X, public Y { . . . };
```

При множественном наследовании возникает проблема неоднозначности из-за совпадающих имен в базовых классах.

Поэтому лучше наследоваться от интерфейсов и классов-контейнеров.

# Deadly Diamond of Death

Проблема ромба [ [wiki](#) ]



если метод класса Child вызывает метод, определенный в классе A, а классы B и C по-своему переопределили этот метод, то от какого класса его наследовать: B или C ?



# Outline

Прошлые темы

Наследование (Inheritance)

Простое наследование

Перегрузка и перекрытие методов

**Динамический полиморфизм**

Виртуальные методы

Ссылки и литература

# Полиморфизм

**Полиморфизм** - обработка разных типов данных одним способом.

**Полиморфизм** - это один интерфейс — много реализаций

Б. Страуструп

# Outline

Прошлые темы

Наследование (Inheritance)

Простое наследование

Перегрузка и перекрытие методов

Динамический полиморфизм

Виртуальные методы

Ссылки и литература

# Виртуальные методы

**Виртуальный метод** - метод, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения.

**Чистый виртуальный (абстрактный) метод** - виртуальный метод для которого не приведена реализация.

# Виртуальные методы

Зачем нужны?

- ▶ Реализует динамический полиморфизм
- ▶ Упрощает интерфейс

У целого набора классов может быть метод с одним именем и набором параметров (или несколько таких), который решает одну и ту же задачу, но специфичным для каждого класса способом. Какая конкретно реализация метода должна быть вызвана определяется во время выполнения программы.

# Раннее и позднее связывание

**Статическая типизация (раннее связывание)** —  
определение типа на этапе компиляции.

**Динамическая типизация (позднее связывание)** —  
определения типа во время выполнения программы.

# Виртуальные методы

- ▶ Рассмотрим нескольких классов с общим базовым классом (на любом уровне)
- ▶ В указатель на базовый класс, можно записать указатель на любой из производных классов
- ▶ Предположим всех классов есть переопределённый метод
- ▶ Если такой метод определён как **виртуальный** (virtual) в базовом классе
- ▶ То на этапе выполнения программы, каждый раз будет определяться тип объекта записанного в указатель на базовый класс
- ▶ После того как тип объекта будет определён, будет вызван соответствующий ему метод

# Виртуальные методы

Механизм **виртуальных методов** заключается в том, что, результат вызова виртуального метода с использованием *указателя* или *ссылки* зависит не от того, на основе какого типа создан указатель, а от типа объекта, на который указывает этот указатель.

Таким образом можно записывать в указатель на базовый класс указатели на производные классы.

При этом тип записанного объекта будет определяться каждый раз на этапе выполнения программы



# Виртуальные методы

```
class Shape{
public:
    // ...
    float virtual area() = 0;    // виртуальный метод без реализации -
                                // это абстрактный метод
};

class Square: public Shape{
    float a;
public:
    // ...
    float virtual area() {return a*a;} };

class Circle: public Shape{
    float r;
public:
    // ...
    float virtual area() {return M_PI * r*r;} };

// Полиморфизм реализуется с использованием указателей на объекты
Shape *s;
Square square = new Square(10);
Circle circle = new Circle(10);
s = square;
cout << s->area() << endl;    // 100
s = circle;
cout << s->area() << endl;    // 314.159
```

# Виртуальные методы

Динамический полиморфизм удобен при работе с набором объектов

```
// набор указателей на базовые классы
vector<Shape*> shapes;

// в shapes можно записать указатель на объект
// любого из производных классов
for (unsigned i =0; i< n; i++){
    if ( rand()%2 == 0 )
        shapes.push_back( new Square(rand()%100) );
    else
        shapes.push_back( new Circle(rand()%100) );
}

// Какой из методов вызывать (для какого класса)
// будет определено на этапе выполнения
for (Shape *s: shapes){
    cout << s->area();
}
```

# Динамическая информация о типе (RTTI)

**Таблица виртуальных методов** (virtual method table, VMT)  
— координирующая таблица или vtable — механизм,  
используемый в языках программирования для поддержки  
динамического соответствия (или метода позднего  
связывания).

# Вопросы

- ▶ Чем отличается переопределение виртуальных методов от переопределения виртуальных?
- ▶ Для чего нужен динамический полиморфизм? Приведите примеры.
- ▶ Как задействовать позднее связывание?

# Outline

Прошлые темы

Наследование (Inheritance)

Простое наследование

Перегрузка и перекрытие методов

Динамический полиморфизм

Виртуальные методы

Ссылки и литература

## Ссылки и литература

1. <https://stepik.org/course/7> Программирование на языке C++
2. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений. 720 с. 2010 г. 700 страниц. Теория. Примеры на C++. Картинки! Вторая половина книги - примеры OOA и OOD с UML диаграммами.
3. MSDN - Microsoft Developer Network
4. Qt 5.X. Профессиональное программирование на C++. Макс Шлее. 2015 и более поздние издания г. 928 с. Книга периодически обновляется с выходом новых версий фреймворка Qt.
5. [www.stackoverflow.com](http://www.stackoverflow.com) - система вопросов и ответов
6. [draw.io](http://draw.io) — создание диаграмм.

# Материалы курса

Слайды, вопросы к экзамену, задания, примеры

[github.com/VetrovSV/OOP](https://github.com/VetrovSV/OOP)

