

# Лекция 3.

## Создание классов. Наследование.

2 семестр

Лектор: ст.пр. Бельченко Ф.М.

## Создание класса

```
#include <iostream>
using namespace std;
// начало объявления класса
class Date // имя класса
{
public: // спецификатор доступа
    void message() // функция (метод класса) выводящая сообщение на экран
    {
        cout << "Example_1\ntheme: Classes and Objects in C + +\n";
    }
}; // конец объявления класса CppStudio

int main(int argc, char* argv[])
{
    Date objDate; // объявление объекта
    objDate.message(); // вызов функции класса message
    system("pause");
    return 0;
}
```

# Конструктор

**Конструктор** — специальная функция, которая выполняет начальную инициализацию элементов данных, причём имя конструктора обязательно должно совпадать с именем класса. Важным отличием конструктора от остальных функций является то, что он не возвращает значений вообще никаких, в том числе и `void`.

В любом классе должен быть конструктор. Даже если явным образом конструктор не объявлен (как в предыдущем классе), то компилятор предоставляет конструктор по умолчанию, без параметров.

## Пример с конструктором

```
#include <iostream>
using namespace std;
class Date // имя класса
{
private: // спецификатор доступа private
    int day, // день
        month, // месяц
        year; // год

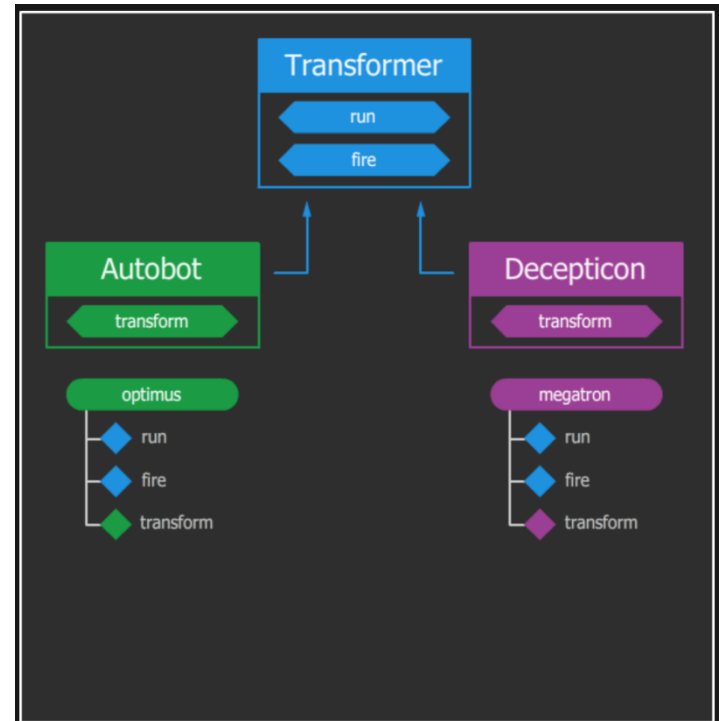
public: // спецификатор доступа public
    Date(int date_day, int date_month, int date_year) // конструктор класса
    {
        setDate(date_day, date_month, date_year); // вызов функции установки даты
    }
    void message() // функция (метод класса) выводящая сообщение на экран
    {
        cout << "\nExample\ntheme: Classes and Objects in C + +\n";
    }
    void setDate(int date_day, int date_month, int date_year) // установка даты в формате дд.мм.гг
    {
        day = date_day; // инициализация день
        month = date_month; // инициализация месяц
        year = date_year; // инициализация год
    }
    void getDate() // отобразить текущую дату
    {
        cout << "date: " << day << "." << month << "." << year << endl;
    }
}; // конец объявления класса CppStudio

int main(int argc, char* argv[])
{
    Date objDate(11, 11, 2011); // объявление объекта и инициализация элементов данных
    objDate.message(); // вызов функции message
    objDate.getDate(); // отобразить дату
    system("pause");
    return 0;
}
```

# Наследование

Новые классы могут быть производными от существующих классов с помощью механизма под названием "наследование".

Классы, используемые для наследования, называются "базовыми классами" определенного производного класса.



## Когда может понадобиться наследование?

В данном случае класс **Employee** фактически содержит функционал класса **Person**: свойства **name** и **age** и функцию **print**. В целях демонстрации все переменные здесь определены как публичные. И здесь, с одной стороны, мы сталкиваемся с повторением функционала в двух классах. С другой стороны, мы также сталкиваемся с отношением **is** ("является"). То есть мы можем сказать, что сотрудник компании **ЯВЛЯЕТСЯ** человеком. Так как сотрудник компании имеет в принципе все те же признаки, что и человек (имя, возраст), а также добавляет какие-то свои (компанию).

```
1  class Person
2  {
3  public:
4      void print() const
5      {
6          std::cout << "Name: " << name << "\tAge: " << age << std::endl;
7      }
8      std::string name;      // имя
9      unsigned age;         // возраст
10 };
11 class Employee
12 {
13 public:
14     void print() const
15     {
16         std::cout << "Name: " << name << "\tAge: " << age << std::endl;
17     }
18     std::string name;      // имя
19     unsigned age;         // возраст
20     std::string company;   // компания
21 };
```

# Наглядный общий пример наследования



```
class Transformer(){ // базовый класс
    function run(){
        // код, отвечающий за бег
    }
    function fire(){
        // код, отвечающий за стрельбу
    }
}

class Autobot(Transformer){ // дочерний класс, наследование от Transformer
    function transform(){
        // код, отвечающий за трансформацию в автомобиль
    }
}

class Decepticon(Transformer){ // дочерний класс, наследование от Transformer
    function transform(){
        // код, отвечающий за трансформацию в самолет
    }
}

optimus = new Autobot()
megatron = new Decepticon()
```



## Теперь конкретный код

```
#include <iostream>

class Person
{
public:
    void print() const
    {
        std::cout << "Name: " << name << "    Age: " << age << std::endl;
    }

    std::string name;    // имя
    unsigned age;        // возраст
};

class Employee : public Person
{
public:
    void print() const
    {
        std::cout << "Name: " << name << "    Age: " << age << "    Company: " << company << std::endl;
    }

    std::string company;    // компания
};

int main()
{
    Person tom;
    tom.name = "Tom";
    tom.age = 23;
    tom.print();    // Name: Tom    Age: 23

    Employee bob;
    bob.name = "Bob";
    bob.age = 31;
    bob.company = "Microsoft";
    bob.print();    // Тут добавится компания
}
```

Консоль отладки Microsoft Visual Studio

Name: Tom	Age: 23	
Name: Bob	Age: 31	Company: Microsoft

C:\Users\phile\source\repos\ClassLec\x64\Debug\ClassLec.exe  
(процесс 15936) завершил работу с кодом 0.  
Нажмите любую клавишу, чтобы закрыть это окно:



## Конструкторы и наследование

Теперь сделаем все переменные приватными, а для их инициализации добавим конструкторы.

И тут стоит учитывать, что конструкторы при наследовании **не наследуются**. И если базовый класс содержит только конструкторы с параметрами, то производный класс должен вызывать в своем конструкторе один из конструкторов базового класса.

Вначале будет вызываться конструктор базового класса Person, в который будут передаваться значения "Bob" и 42. И таким образом будут установлены имя и возраст. Затем будет выполняться собственно конструктор Employee, который установит компанию.

# Пример наследования с конструкторами

```
#include <iostream>

class Person
{
public:
    Person(std::string name, unsigned age)
    {
        this->name = name;
        this->age = age;
    }
    void print() const
    {
        std::cout << "Name: " << name << "\tAge: " << age << std::endl;
    }
private:
    std::string name;    // имя
    unsigned age;        // возраст
};

class Employee : public Person
{
public:
    Employee(std::string name, unsigned age, std::string company) : Person(name, age)
    {
        this->company = company;
    }
private:
    std::string company; // компания
};

int main()
{
    Person person{ "Tom", 38 };
    person.print();    // Name: Tom    Age: 38

    Employee employee{ "Bob", 42, "Microsoft" };
    employee.print();  // Name: Bob    Age: 42
}
```

Консоль отладки Microsoft Visual Studio

Name: Bob Age: 42

C:\Users\phile\source\repos\ClassLec\x64\Debug\ClassLec.exe  
(процесс 7160) завершил работу с кодом 0.

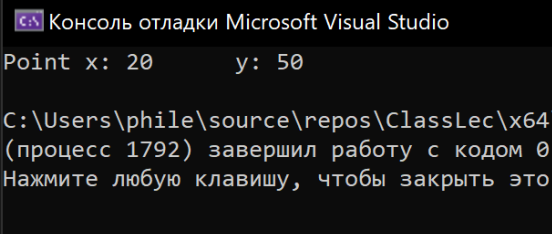
Нажмите любую клавишу, чтобы закрыть это окно:

# Ключевое слово **this**

```
#include <iostream>

class Point
{
public:
    Point(int x, int y)
    {
        this->x = x;
        this->y = y;
    }
    void showCoords()
    {
        std::cout << "Point x: " << this->x << "\t y: " << y << std::endl;
    }
private:
    int x;
    int y;
};

int main()
{
    Point p1{ 20, 50 };
    p1.showCoords();
}
```



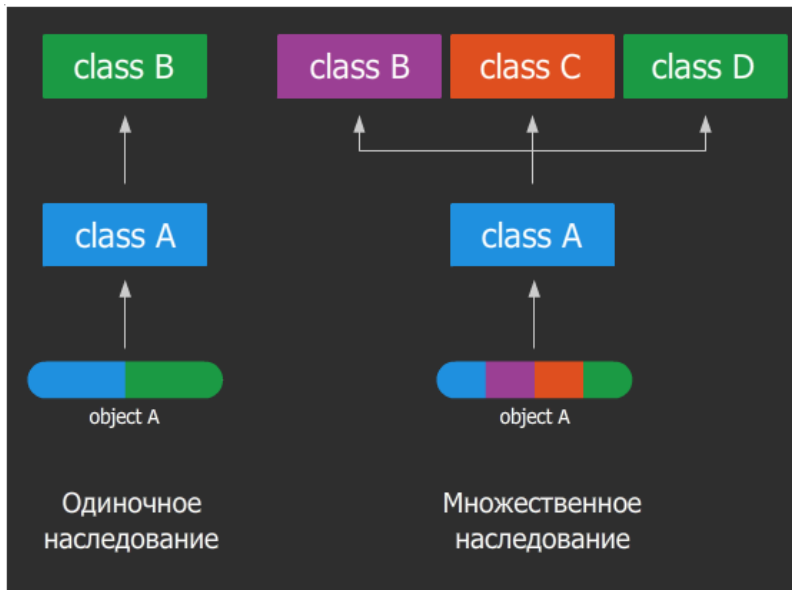
В данном случае определен класс Point, который представляет точку на плоскости. И для хранения координат точки в классе определены переменные x и y.

Ключевое слово **this** представляет указатель на текущий объект данного класса. Соответственно через **this** мы можем обращаться внутри класса к любым его членам.

Для обращения к переменным используется указатель **this**. Причем после **this** ставится не точка, а стрелка ->.

В большинстве случаев для обращения к членам класса вряд ли понадобится ключевое слово **this**. Но оно может быть необходимо, если параметры функции или переменные, которые определяются внутри функции, называются также как и переменные класса. К примеру, чтобы в конструкторе разграничить параметры и переменные класса как раз и используется указатель **this**.

# Множественное наследование



Производный класс может иметь несколько прямых базовых классов. Подобный тип наследования называется **множественным наследованием** в отличие от одиночного наследования, при котором используется один базовый класс. Поскольку это несколько усложняет иерархию наследования, то используется гораздо реже, чем одиночное наследование.

## Пример с множественным наследованием

```
#include <iostream>

class Camera    // класс фотокамеры
{
public:
    void makePhoto()
    {
        std::cout << "making photo" << std::endl;
    }
};

class Phone     // класс телефона
{
public:
    void makeCall()
    {
        std::cout << "making call" << std::endl;
    }
};

// класс смартфона
class Smartphone : public Phone, public Camera
{
};

int main()
{
    Smartphone iphone;
    iphone.makePhoto();    // сделать фото
    iphone.makeCall();     // сделать вызов
}
```

Консоль отладки Microsoft Visual Studio

making photo  
making call

C:\Users\phile\source\repos\ClassLec\x64\Debug\ClassLec.exe  
(процесс 17448) завершил работу с кодом 0.  
Нажмите любую клавишу, чтобы закрыть это окно:

# Проблемы декомпозиции задач, усложняющие реализацию наследований

Любопытно, что чрезмерно глубокая иерархия наследования может привести к обратному эффекту — усложнению при попытке разобраться, кто от кого наследуется, и какой метод в каком случае вызывается.

К тому же, не все архитектурные требования можно реализовать с помощью наследования. Поэтому применять наследование следует без фанатизма.



НЕ УСЛОЖНЯЙ - главное правило ООП)))

# Запрет наследования

Иногда наследование от класса может быть нежелательно. И с помощью спецификатора **final** мы можем запретить наследование:

```
797 #include <iostream>
798
799 class Person final
800 {
801 public:
802     Person(std::string name, unsigned age)
803     {
804         this->name = name;
805         this->age = age;
806     }
807     void print() const
808     {
809         std::cout << "Name: " << name << "\tAge: " << age << std::endl;
810     }
811 private:
812     std::string name;    // имя
813     unsigned age;       // возраст
814 };
815
```

81 % 6 0 ↑ ↓ ◀ ▶ Стр: 799 Симв: 19 Пробелы CR

Вывод

Показать выходные данные из: Сборка

Сборка начата...

```
1>----- Сборка начата: проект: ClassLec, Конфигурация: Debug x64 -----
1>ClassLec.cpp
1>C:\Users\phile\source\repos\ClassLec\ClassLec\ClassLec.cpp(817,1): error C3246: "Employee": нельзя наследовать из "Person", так как было объявление в качестве "final"
1>C:\Users\phile\source\repos\ClassLec\ClassLec\ClassLec.cpp(799): message : см. объявление "Person"
1>Сборка проекта "ClassLec.vcxproj" завершена с ошибкой.
===== Сборка: успешно: 0, с ошибками: 1, без изменений: 0, пропущено: 0 =====
```