

# Junior C++ developer

## Лекция 4

- Пользовательские типы данных
- Конструктор/деструктор
- Список инициализации

В C++ традиционно каждый класс описывается в отдельном файле.

Для каждого класса создается файл-заголовочник и файл с исходным кодом.

Заголовочный файл не содержит в себе реализации, он только “обещает” компилятору, что где-то в нашей программе есть реализация.

Заголовочный файлы  
имеют расширение **.h**  
или **.hpp**  
Файлы с исходным  
кодом **.cpp**

# Классы и объекты

Файл triangle.h

```
class Triangle
{
private:
    double a;
    double b;
    double c;

public:
    Triangle(double a_in, double b_in, double c_in);
    const double perimeter();
    void scale(double s);
};
```

## Ключевое слово



```
class Triangle
```

```
{
```

```
private:
```

```
    double a;
```

```
    double b;
```

```
    double c;
```

```
public:
```

```
    Triangle(double a_in, double b_in, double c_in);
```

```
    double perimeter() const;
```

```
    void scale(double s);
```

```
};
```

Имя класса



```
class Triangle
{
    private:
        double a;
        double b;
        double c;

    public:
        Triangle(double a_in, double b_in, double c_in);
        double perimeter() const;
        void scale(double s);
};
```

## Модификаторы доступа

```
class Triangle
```

```
{
```

```
private:
```

```
    double a;
```

```
    double b;
```

```
    double c;
```

```
public:
```

```
    Triangle(double a_in, double b_in, double c_in);
```

```
    double perimeter() const;
```

```
    void scale(double s);
```

```
};
```

```
class Triangle
```

```
{
```

```
private:
```

```
double a;
```

```
double b;
```

```
double c;
```

Поля класса



```
public:
```

```
Triangle(double a_in, double b_in, double c_in);
```

```
double perimeter() const;
```

```
void scale(double s);
```

```
};
```

!!!

С++ (как язык) не требует чтобы все поля класса были private, но правила хорошего тона программирования требуют этого.

```
class Triangle
```

```
{
```

```
    private:
```

```
        double a;
```

```
        double b;
```

```
        double c;
```

```
    public:
```

```
        Triangle(double a_in, double b_in, double c_in);
```

```
        double perimeter() const ;
```

```
        void scale(double s);
```

```
};
```

Методы  
класса


```
graph LR; A[Методы класса] --> B[Triangle(double a_in, double b_in, double c_in);]; A --> C[double perimeter() const ;]; A --> D[void scale(double s);];
```



```
#include "triangle.h"
```

```
double Triangle::perimeter() const  
{  
    return a + b + c;  
}
```

Подключение  
заголовочного  
файла



```
#include "triangle.h"
```

```
double Triangle::perimeter() const  
{  
    return a + b + c;  
}
```

Реализация  
метода класса  
***Triangle***

```
#include "triangle.h"
```

```
double Triangle::perimeter() const  
{  
    return a + b + c;  
}
```

```
#include "triangle.h"
```

```
double Triangle::perimeter() const  
{  
    return a + b + c;  
}
```

Обращение к  
полям класса



# Практика

1. Написать объявление класса Person, содержащего:
  - а. Поля:
    - i. Имя
    - ii. Фамилия
    - iii. Уникальный идентификатор
  - б. Метод
    - i. Возвращающий значение поля имя

# Практика

```
#include <string>
class Person
{
public:
    std::string getName();
private:
    std::string first_name;
    std::string last_name;
    int unic_id;
};
```

1. Написать объявление класса Person, содержащего:

а. Поля:

i. Имя

ii. Фамилия

iii. Уникальный идентификатор

б. Метод

i. Возвращающий полное имя

2. Написать реализацию класса

# Практика

```
#include "person.h"
```

```
std::string Person::getName()  
{  
    return first_name +  
        " " + last_name;  
}
```

1. Написать объявление класса Person, содержащего:

а. Поля:

i. Имя

ii. Фамилия

iii. Уникальный идентификатор

б. Метод

i. Возвращающий полное имя

2. Написать реализацию класса

3. В функции main создать 2 экземпляра класса

# Практика

```
#include "person.h"
int main ()
{
    Person p1;
    Person p2;
    return 0;
}
```

!¿Что будет

1. Чему будет равна переменная ***name***

```
std::name = p1.getName();
```

2. Чему будет равна ***i***?

```
int i = p1.unic_id;
```



# Модификаторы доступа

- Все поля и методы класса имеют модификаторы доступа
  - **Public**: Может быть вызван из любого места в программе.
  - **Private**: Может использоваться только внутри класса.

# Объект

Каждый объект имеет свою копию полей класса в памяти

Объектом называется экземпляр класса

(instance of a class)

- Конкретный объект в памяти
- Имеют собственную копию полей класса
- Единственный и уникальный представитель класса
- Инициализируется конструктором



Класс



Экземпляры класса

# Инициализация полей по умолчанию

- Если поля класса явно не инициализируются они инициализируются значениями по умолчанию

# Конструктор

- имя конструктора совпадает с именем класса
- не имеет типа возвращаемого значения
- создает экземпляр класса
- инициализирует переменные объекта класса



# Практика

1. Написать конструктор класса Person:
  - a. Объявление в заголовочном файле
  - b. Реализация в файле .cpp

# Практика

## 1. Написать конструктор класса Person:

а. Объявление в заголовочном файле

```
#include <string>
class Person
{
public:
    Person(std::string f_name, std::string l_name, int id);

    std::string getName();
private:
    std::string first_name;
    std::string last_name;
    int unic_id;
};
```

Конструктор, который не принимает параметров называется **конструктор по умолчанию**

```
Person();
```

# Практика

## 1. Написать конструктор класса Person:

- Объявление в заголовочном файле
- Реализация в файле .cpp

```
#include "person.h"
```

```
Person::Person(std::string f_name, std::string l_name, int id) :
```

```
    first_name(f_name),
```

```
    last_name(l_name),
```

```
    unic_id(id)
```

```
{
```

```
}
```

```
std::string Person::getName()
```

```
{
```

```
    return first_name + " " + last_name;
```

```
}
```



# Список инициализации

```
Person::Person(std::string f_name,  
               std::string l_name,  
               int id) :  
    first_name(f_name),  
    last_name(l_name),  
    unic_id(id)  
{  
}
```

Класс Person имеет три поля:  
`first_name`, `last_name`, `unic_id`.

Конструктор принимает три параметра:

*std::string f\_name*, *std::string l\_name*, *int id*

*first\_name(f\_name)* - полю класса *first\_name* устанавливается значение *f\_name*

*last\_name(l\_name)* - полю *last\_name* выставляется значение *l\_name*

*unic\_id(id)* - поле *unic\_id* становится равно значению *id*

# Список инициализации

Зачем использовать список инициализации если можно инициализировать поля в теле конструктора?

## Список инициализации

```
Person::Person(std::string f_name,  
               std::string l_name,  
               int id) :  
    first_name(f_name),  
    last_name(l_name),  
    unic_id(id)  
{  
}
```

VS

## Инициализация в теле конструктора

```
Person::Person(std::string f_name,  
               std::string l_name,  
               int id)  
{  
    first_name = f_name;  
    last_name = l_name;  
    unic_id = id;  
}
```

# Список инициализации

На самом деле все переменные создаются ДО момента когда начинается выполнение тела конструктора.

Что это значит?

В C++ у всех пользовательских типов данных, кроме встроенных (int, double, float, double, char, bool), имеется собственные конструкторы, которые вызываются в момент создания объекта.

То есть, для переменной типа `std::string` в момент попадания в тело конструктора, уже был вызвал конструктор по умолчанию и переменная уже проинициализирована пустой строкой, и когда мы ей задаем значение в теле, то происходит двойная работа, мы задаем значение переменной дважды.

В списке инициализации происходит создание объекта сразу с правильным значением.

# Список инициализации

Зачем использовать список инициализации если можно инициализировать поля в теле конструктора?

Кратко :

Избавляет программу от лишней работы, сразу создает объект с нужным значением.

# Область видимости

## Время жизни

- Конструктор
- Область видимости
- Деструктор

## RAII

Resource Acquisition In Initialization (RAII) - Получение ресурса есть инициализация

- Получение ресурса в конструкторе
- Освобождение в деструкторе

# Деструктор

- имя деструктора совпадает с именем класса, перед именем знак ~
- не имеет типа возвращаемого значения
- не имеет входных параметров
- класс может иметь только один деструктор
- выполняет освобождение использованных объектом ресурсов и удаление нестатических переменных объекта.

# Практика

```
#include <string>
```

```
class Person
```

```
{
```

```
public:
```

```
    Person(std::string f_name, std::string l_name, int id);
```

```
    Person();
```

```
    ~Person();
```

```
    std::string getName();
```

```
private:
```

```
    std::string first_name;
```

```
    std::string last_name;
```

```
    int unic_id;
```

```
};
```

# Практика

```
Person::Person(std::string f_name, std::string l_name, int id) :
    first_name(f_name),
    last_name(l_name),
    unic_id(id)
{
    std::cout << "Constructing " <<
    first_name << " " << last_name << std::endl;
}
Person::Person() :
    unic_id(0)
{
    std::cout << "Constructing " <<
    first_name << " " << last_name << std::endl;
}

Person::~~Person()
{
    std::cout << "Destructing " <<
    first_name << " " << last_name << std::endl;
}

std::string Person::getName()
{
    return first_name + " " + last_name;
}
```



# Практика

```
#include <iostream>
#include "person.h"
```

```
int main()
{
    Person p1("Name", "LastName", 1);
    {
        Person p2;
    }

    std::cout << p1.getName() << std::endl;

    return 0;
}
```

## Output:

```
Constructing Name LastName
Constructing
Destructing
Name LastName
Destructing Name LastName
```

# struct vs. class

**Единственное** различие между структурами и классами в С++ это уровень доступа по умолчанию

- struct – public по умолчанию
- class – private по умолчанию

```
struct Triangle {  
    double a;  
    double b;  
    double c;  
    ...  
};
```

a, b, c  
public

```
class Triangle {  
    double a;  
    double b;  
    double c;  
    ...  
};
```

a, b, c  
private

Тем не менее  
классы и  
структуры  
используются в  
С++ для разных  
целей