# Основы ООП в C++

Артамонов Ю.Н.

Филиал «Котельники» университета «Дубна»

10 октября 2018 г.

# Содержание

# Базовые понятия ООП

Человек мыслит объектами: люди, животные, самолеты, дома. Куда бы мы не посмотрели, мы везде видим объекты. При этом человеческое мышление обладает способностью к абстракции: выделению объектов на экране монитора, а не светящихся пикселей, жилых домов, а не груды кирпичей и т.д. Объекты естественным образом выстраиваются в определенную иерархию по уровню абстракции от конкретных свойств: живое существо - человек - мужчина - Сидоров. Каждый объект в нашем понимании обладает определенным набором свойств (атрибутов): у человека - это имя, цвет волос и т.д., дальнейшие уточнения атрибутов приводят к все более конкретному объекту. Кроме атрибутов, объекты обладают различным поведением: человек ходит, разговаривает и т.д.

*Объектно-ориентированное программирование* моделирует объекты реального мира при помощи их программных эквивалентов.

# Базовые понятия ООП

ООП использует понятие *класса*, когда ряд объектов имеет одинаковые атрибуты. Также оно использует отношение *наследования* или даже *сложного наследования*, когда вновь создаваемые классы объектов получаются путем наследования характеристик существующих классов, но при этом содержат и свои собственные уникальные атрибуты, поведение. Это подобно тому, как дети наследуют свойства своих родителей, но у них всегда есть своя индивидуальность. Также как люди посылают друг другу сообщения, так и *объекты взаимодействуют посредством сообщений*. ООП скрывает - *инкапсулирует* данные (атрибуты) и функции (поведение) внутри класса. Это означает, что узнать о значениях скрытых данных, функций другие объекты могут только, если в классе реализован специальный интерфейс для доступа к ним - по принципу: можно эффективно управлять автомобилем, не зная его устройство.

# Базовые понятия ООП

Инкапсуляция позволяет программисту не задумываться над внутренней реализацией кода, что существенно экономит время. Действительно, в интерфейсе класса осуществляются необходимые проверки на допустимость значений атрибутов, функций, и программист не задумывается о корректности этих значений, не погружается в изучение чужого кода.
При этом ООП является дальнейшим развитием понятия структуры.
Для демонстрации этого реализуем структуру Time:

```
struct Time{
  int hour;
  int minute;
  int second;
}
int main(){
  Time t;
  ...
}
```

# Особенности и недостатки структур

При доступе к полям структуры нет никакой проверки на допустимость вводимых значений. Например, пользователь может ввести t.hour = 77; - что некорректно. Это происходит потому, что программист непосредственно манипулирует типом данных. Не сущесвует никакого интерфейса, позволяющего гарантировать, что программист корректно использует типы данных, и что данные остаются действительными. Рассмотрим реализацию данной структуры в ООП.

# Пример реализации класса Time

```
#include<iostream>
using namespace std;
class Time
{
public:
  Time();
  void setTime(int, int, int);
  void printTime();
private:
  int hour;
  int minute;
  int second;
};
```

# Пример реализации класса Time (продолжение)

```
Time::Time() {hour = minute = second = 0;}
void Time::setTime(int h, int m, int s)
{
  hour = (h>=0 && h<24)? h : 0;
  minute = (m>=0 && m<60)? m : 0;
  second = (s>=0 && s<60)? s : 0;
}
void Time::printTime()
{
  cout<< ((hour == 0 || hour == 12) ? 12 : hour%12)
      <<":"<< (minute<10 ? "0" : "") << minute
      <<":"<<(second<10 ? "0" :"")<<second
      <<(hour<12 ? "AM" : "PM");
}
```

# Пример реализации класса Time (продолжение)

```
int main()
{
  Time t;
  cout<<"Начальное значение t"<<endl;
  t.printTime(); cout<<endl;
  t.setTime(14, 63, 17);
  cout<<"Изменили значение"<<endl;
  t.printTime(); cout<<endl;
  return 0;
}
```

# Пояснения к коду

Определение класса Time начинается с ключевого слова class. Тело определения класса ограничено левой и правой фигурными скобками. Определение класса завершается точкрй с запятой. В определении класса, аналогично структуре Time, содержатся три целочисленных элемента: hour, minute, second. Метки public:, private: называются *спецификаторами доступа к элементам*. Все элементы данных и функций, объявленные после спецификатора public: (вплоть до следующего спецификатора доступа) доступны всюду, где программа имеет доступ к какому-либо объекту класса Time. Все элементы данных и функций, объявленные после спецификатора private (вплоть до следующего спецификатора доступа), доступны только для функций и данных класса. Спецификаторы доступа к элементам класса всегда заканчиваются двоеточием (:) и могут многократно появляться в определении класса.

# Пояснения к коду (продолжение)

Определение класса после спецификатора public: содержит прототипы трех функций - *методов класса*: Time, setTime, printTime. Эти функции являются открытыми (публичными) и реализуют интерфейс класса, они могут использоваться для доступа и манипулирования данными класса. Функция с тем же именем, что и сам класс, называется *конструктором класса*. Конструктор - это специальная функция класса, которая инициализирует элементы данных при создании объекта (*экземпляра класса*).

Три целочисленных элемента появляются после спецификатора private:. Это означает, что эти элементы доступны только для функций класса. Именно в этом смысле говорят, что реализация класса скрыта от пользователей класса.

Имя класса становится новым типом данных. Может существовать много объектов класса, подобно тому, как много может быть переменных типа int.

# Пояснения к коду (продолжение)

Определение класса содержит объявление его элементов: данных и функций. Объявления функций представляют собой прототипы функций. Функции могут быть определены внутри класса (вместо включения туда прототипа), однако хорошим стилем считается определение функций вне определения класса. Для доступа к функциям класса во внешней программе для их определения используется *операция разрешения области действия* (::), поскольку различные классы могут иметь одинаковые имена элементов. Операция разрешения области действия однозначно привязывает имя элемента к имени класса, однозначно идентифицируя функции данного класса.