# ООП

Мы используем классы чтобы разделить сложную программу на небольшие части, и реализовать каждую эту часть по отдельности

# Создание класса

```
class /*имя класса*/
{
  private:
  /* список свойств и методов для использования внутри класса */
  public:
  /* список методов доступных другим функциям и объектам программы */
  protected:
  /*список средств, доступных при наследовании*/
};
```

После того как мы написали содержимое класса, мы сможем его использовать, создавая переменные этого типа (объекты).

Сам по себе класс ничего не делает, **это просто тип данных**, который мы создали.

```cpp
int main()
{
    setlocale(0, "");
    Animal  slon_Vasia, cherepaha_Anna;
    //далее присваиваем значения всем полям 2х объектов
    slon_Vasia.animal_type = "Слон";
    slon_Vasia.name = "Вася";
    slon_Vasia.sound = "УУУУУуууу";
    cherepaha_Anna.animal_type = "Черепаха";
    cherepaha_Anna.name = "Аня";
    cherepaha_Anna.sound = "буль-буль";

    //вызываем их методы say
    slon_Vasia.say();
    cherepaha_Anna.say();

}
```

```cpp
class Animal {
public:
    int age;
    string animal_type;
    string name;
    string sound;

    void say() {
        cout << animal_type << ' ' << name << " говорит " << sound;
    }
};
```

Как мы видим из предыдущего примера, задавать все значения вручную неудобно и долго, и также для этого они должны быть *Public*, что нарушает суть ООП

Поэтому для того чтобы задавать начальные значения существуют **конструкторы**

```cpp
class Animal {
public:
    int age;
    string animal_type;
    string name;
    string sound;

    //конструктор, всегда называется так же как и класс, их может быть несколько разных
    //с разными параметрами, но название у всех одно - ИМЯ КЛАССА
    Animal(int new_age, string new_type, string new_name, string new_sound) {
        age = new_age;
        animal_type = new_type;
        name = new_name;
        sound = new_sound;
    }

    void say() {
        cout << animal_type << ' ' << name << " говорит " << sound<<endl;
    }
};
```

# Теперь с конструктором мы можем намного удобнее создавать объекты классов, сравните:

## Вручную

```
int main()
{
    setlocale(0, "");
    Animal  slon_Vasia, cherepaha_Anna;
    //далее присваиваем значения всем полям 2х объектов
    slon_Vasia.animal_type = "Слон";
    slon_Vasia.name = "Вася";
    slon_Vasia.sound = "УУУУУуууу";
    cherepaha_Anna.animal_type = "Черепаха";
    cherepaha_Anna.name = "Аня";
    cherepaha_Anna.sound = "буль-буль";

    //вызываем их методы say
    slon_Vasia.say();
    cherepaha_Anna.say();

}
```

## С помощью конструктора

```
int main()
{
    setlocale(0, "");
    //Все значения мы задаем через конструктор
    Animal  slon_Vasia(10, "Слон", "Вася", "УУУУУуууу");
    Animal cherepaha_Anna(55, "Черепаха", "Аня", "буль-буль");

    //вызываем их методы say
    slon_Vasia.say();
    cherepaha_Anna.say();
}
```

Когда образуется сложная структура классов, то чтобы не переписывать все одни и те же вещи из одного класса в другой  и как-то связать классы друг с другом, применяется наследование (потомок наследует все из класса родителя)

```cpp
class Dog :Animal {
public:
    string poroda;

    Dog(int new_age, string new_name, string new_poroda)
        :Animal(new_age, "Собака", new_name, "Гав")
    //передаем конструктору родителю основные значения, чтобы не писать
    // то же самое второй раз
    {
        poroda = new_poroda;
    }

    void say() {
        Animal::say();
        cout<< "Его порода " << poroda<<endl;
    }
};
```

```cpp
int main()
{
    setlocale(0, "");
    //Назначаем начальные значения в конструкторе
    Dog Alex(15, "Алекс", "лайка");
    Alex.say();
```

Консоль отладки Microsoft Visual Studio

```
Собака Алекс говорит Гав
Его порода лайка
```

# И последнее из самого важного:

- В более новых языках программирования(Java, C#,) отказались от наследования нескольких классов ради большей безопасности программ, и в дополнение к классам появились интерфейсы, которых можно наследовать сколько угодно, но они сами по себе не обладают никаким функционалом(в C++ их задачу выполняют виртуальные классы(virtual))

- Интерфейсы позволяют контролировать, что класс умеет выполнять какую-то функцию, чтобы потом это проверять и использовать в программе

# Создали интерфейс, а потом унаследовали от него несколько классов

```
interface Messenger
{
    void sendMessage();
    void getMessage();
}
```

```
class Telegram:Messenger
{
    public void sendMessage()
    {
        System.Console.Write("Отправляем сообщение в Telegram!");
    }
    public void getMessage()
    {
        System.Console.Write("Читаем сообщение в Telegram!");
    }
}
```

```
class Vk : Messenger
{
    public void sendMessage()
    {
        System.Console.Write("Отправляем сообщение в Vk!");
    }
    public void getMessage()
    {
        System.Console.Write("Читаем сообщение в Vk!");
    }
}
```

```
class WhatsApp : Messenger
{
    public void sendMessage()
    {
        System.Console.Write("Отправляем сообщение в WhatsApp!");
    }
    public void getMessage()
    {
        System.Console.Write("Читаем сообщение в WhatsApp!");
    }
}
```

Теперь во избежание неоднозначности мы можем проверять объекты на наследование этого интерфейса , или хранить любые такие объекты в переменной типа интерфейса (Messenger)

```
class Program
{
    static void Main(string[] args)
    {
        //объявляем 3 типа месседжера
        Telegram telegram=new Telegram();
        WhatsApp whatsApp=new WhatsApp();
        Vk vk=new Vk();

        //создаем пользователей, которые могут использовать разные месседжеры
        User Artem=new User(telegram);
        User Pavel=new User(vk);
        User Sveta=new User(whatsApp);
        User Nikita = new User(vk);
    }
}
```

```
class User
{
    private Messenger messenger;

    public User(Messenger new_m)
    {
        messenger = new_m;
    }
}
```

# Немного про сжатие данных

2 основных способа сжатия данных:

- Кодирование нескольких одинаковых символов подряд с помощью счетчика повторений

  AAAABBCAAAAAAAAAA->4A2BC10A

- Замена символов, которые встречаются чаще других более короткими кодами

# Про второй способ:

- Как вы знаете, в любом алфавите некоторые буквы или слова встречаются намного чаще, чем другие. А значит самые частые символы можно кодировать короткими кодами, а редкие длинными
- Например, в коде символов ascii все символы кодируются 8 битным двоичным числом, буква **“о”**- числом **10101110**, и буква **“ё”** числом **11110001**, то есть одинаковой длины, хотя букву “о” мы пишем по статистике почти в 300 раз чаще, чем “ё”

- Тогда чтобы сжать любой текст на русском языке, мы могли бы изменить коды наших символов, и кодировать частую **“о”** коротким кодом **001**, а редкий символ **“ё”** более длинным, например **000000000001**
- Но из-за того что “ё” почти не встречается в тексте, то и объем текста сильно уменьшится

# Сравните код ascii и наш сжимающий код на примере “ооооё”:

- Код ascii: **10101110101011101010111010101110111110001**
- Наш код: **00100100100100000000000001**

# И поподробнее про первый способ (замена подряд идущих символов или значений):

- Так как числа тоже являются частью текста, то мы не можем просто так заменять AAAAA на 5A, ведь “5” тоже  может встретиться в тексте, и мы не сможем понять что это, просто число или длина нашей последовательности
- Значит нам нужно как-то обозначать, то что следующее число будет обозначать именно длину последовательности. Можно взять для этого какой-либо неиспользуемый символ, например |. Тогда при разархивации мы будем знать, что |5A это именно наша длина, но к сожалению немного увеличим объем заархивированного файла

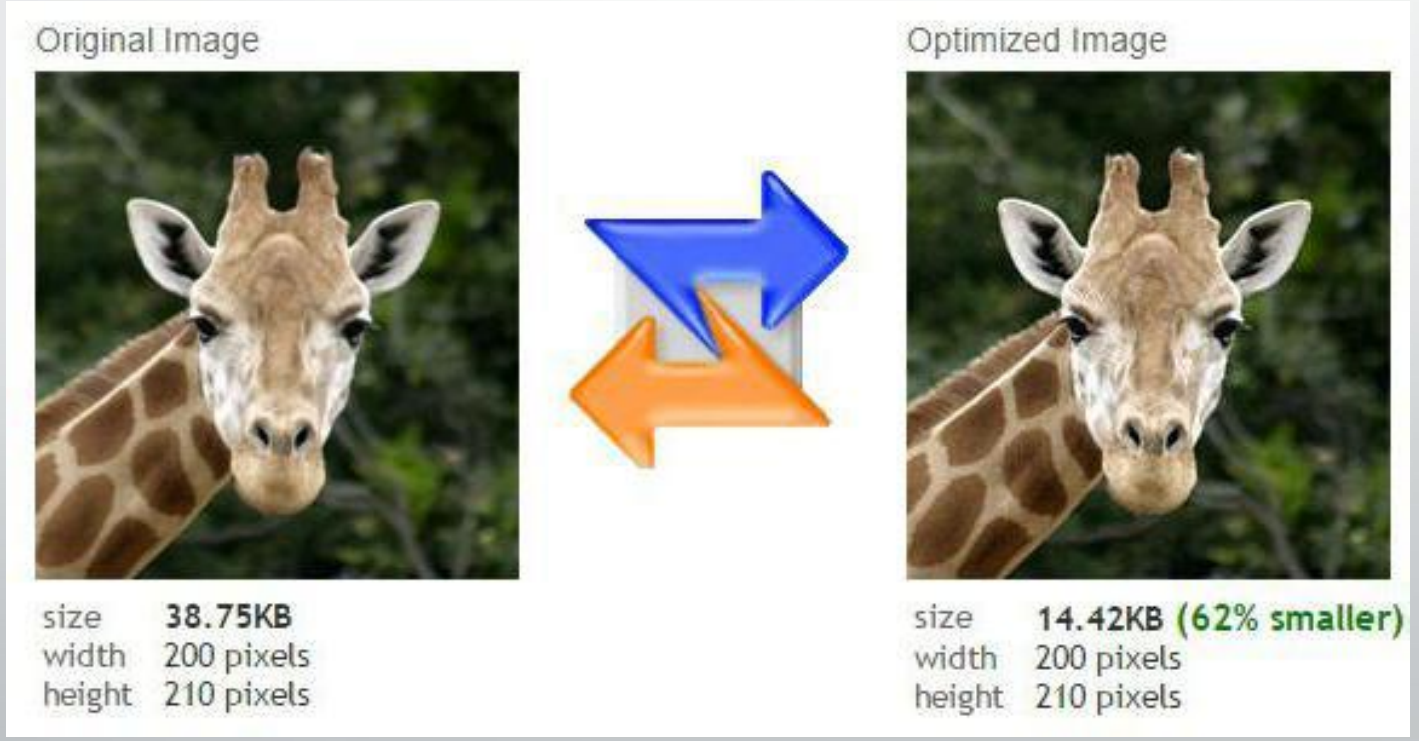
Также существуют алгоритмы сжатия с потерями, в основном для фотографий, звука и видео

- Они сжимают данные так, чтобы изменения не были заметны для человеческих органов чувств, но при этом заметно изменяются в размере
- Как пример это всеми используемые форматы JPEG, Djvu, MP3, Opus

# Сжатие с разной глубиной квантования цветоразностных сигналов

| 256 уровней | 128 уровней | 64 уровня |
|---|---|---|
|  |  |  |
| **32 уровня** | **16 уровней** | **8 уровней** |
|  |  |  |

Для сжатия с потерями существует неисчислимое количество разных алгоритмов, поэтому их рассматривать слишком долго и сложно, посмотрим просто несколько примеров



# На этом все

## JPEG compression comparison

89k

12k