

# ООП

Кафедра ИВТ и ПМ

2018

# План

Абстрактный тип данных

Введение в ООП

Декомпозиция

Принципы ООП

Отношения между классами и UML диаграммы

# Outline

Абстрактный тип данных

Введение в ООП

Декомпозиция

Принципы ООП

Отношения между классами и UML диаграммы

# Абстрактный тип данных

**Абстрактный тип данных (АТД, Abstract Data Type - ADT)** — это математическая модель для типов данных, где тип данных определяется поведением (семантикой) с точки зрения пользователя данных, а именно в терминах возможных значений, возможных операций над данными этого типа и поведения этих операций.

# Абстрактный тип данных

**АТД** – это такой тип данных, который скрывает свою внутреннюю реализацию от клиентов.

Удивительно то, что путем применения абстракции АТД позволяет нам не задумываться над низкоуровневыми деталями реализации, а работать с высокоуровневой сущностью реального мира – Стив Макконнелл.

# Абстрактный тип данных

**АТД** позволяет описать тип данных независимо от языка программирования.

# Шаблон описание абстрактного типа данных

## ADT НаименованиеАбстрактногоТипаДанных

- ▶ **Данные**

... перечисление данных ...

- ▶ **Операции**

- ▶ **Конструктор**

Начальные значения:

Процесс:

- ▶ **Операция. . .**

Вход:

Предусловия:

Процесс:

Выход:

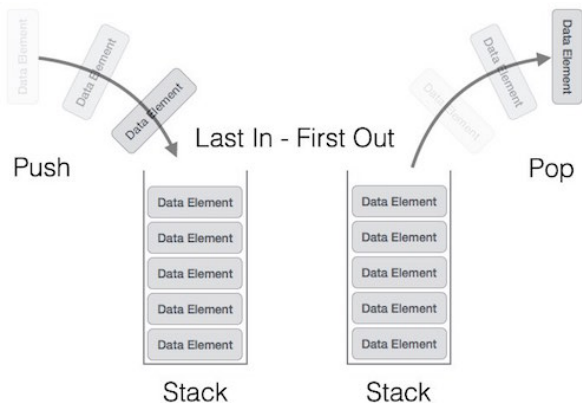
Постусловия:

- ▶ **Операция. . . . .**

Конец ADT НаименованиеАбстрактногоТипаДанных

# Абстрактный тип данных. Пример - стек

**Стек** - абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»)





# Пример ADT - Стек

## ADT Stack

### ► Данные

Список элементов с позицией top, указывающей на вершину стека.

### ► Операции

#### ► Конструктор

Начальные значения:

Нет

Процесс: Инициализация вершины стека.

#### ► StackEmpty

Вход: Нет

Предусловия: Нет

Процесс: Проверка, пустой ли стек

Выход: Возвращать True, если стек пустой, иначе возвращать False.

Постусловия: Нет

# Пример ADT - Стэк

- ▶ Операции (продолжение)

- ▶ **Pop**

- Вход: Нет

- Предусловия: Стэк не пустой

- Процесс: Удаление элемента из вершины стека

- Выход: Возвращает элемент из вершины стека

- Постусловия: Элемент удаляется из вершины стека

- ▶ **Push**

- Вход: Элемент для стека

- Предусловия: Нет

- Процесс: Сохранение элемента в вершине стека

- Выход: Нет

- Постусловия: Стэк имеет новый элемент в вершине

# Пример ADT - Стэк

- ▶ Операции (продолжение)

- ▶ **Peek**

- Вход: Нет

- Предусловия: Стэк не пустой

- Процесс: Нахождение значения элемента в вершине стека

- Выход: Возвращать значение элемента в вершине стека

- Постусловия: Стэк неизменный

- ▶ **ClearStack**

- Вход: Нет

- Предусловия: Нет

- Процесс: Удаление всех элементов из стека и  
переустановка вершины стека

- Выход: Нет

- Постусловия: Стэк переустановлен в начальные условия

Конец ADT Stack

## Чем полезен АДТ?

- ▶ Инкапсуляция деталей реализации. Это означает, что единожды инкапсулировав детали реализации работы АДТ мы предоставляем клиенту интерфейс, при помощи которого он может взаимодействовать с АДТ. Изменив детали реализации, представление клиентов о работе АДТ не изменится.
- ▶ Снижение сложности. Путем абстрагирования от деталей реализации, мы сосредотачиваемся на интерфейсе, т.е на том, что может делать АДТ, а не на том как это делается. Более того, АДТ позволяет нам работать с сущностью реального мира.
- ▶ Ограничение области использования данных. Используя АДТ мы можем быть уверены, что данные, представляющие внутреннюю структуру АДТ не будут зависеть от других участков кода. При этом реализуется “независимость” АДТ.
- ▶ Высокая информативность интерфейса. АДТ позволяет представить весь интерфейс в терминах сущностной предметной области, что, согласитесь, повышает удобочитаемость и информативность интерфейса.

# Outline

Абстрактный тип данных

Введение в ООП

Декомпозиция

Принципы ООП

Отношения между классами и UML диаграммы

# Структурное программирование

- ▶ Что такое структурное программирование?
- ▶ Что такое процедурное программирование?
- ▶ Что такое модульное программирование?

# Outline

Абстрактный тип данных

Введение в ООП

Декомпозиция

Принципы ООП

Отношения между классами и UML диаграммы

# Декомпозиция

- ▶ Большие программы создавать сложно
- ▶ Декомпозиция - разделение предметной области, задач и исходного кода на части - упрощает разработку.



- ▶ **Алгоритмическая декомпозиция** используется в структурном программировании.
  - ▶ Задачи разбиваются на подзадачи
  - ▶ Решение задачи и сама программа - *процесс*
  - ▶ Структуры данных - вторичны

Однако мир представляет собой совокупность взаимодействующих объектов...

- ▶ **Объектно-ориентированная декомпозиция.**
  - ▶ Предметная область представляется разбивается на объекты
  - ▶ Задача представляется как взаимодействие отдельных объектов.

*"Вместо процессоров, бесцеремонно расхватывающих структуры данных, мы имеем дело с благонаправными объектами, вежливо просящими друг друга об услугах."*

– Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений

Примеры объектов?

# Outline

Абстрактный тип данных

Введение в ООП

Декомпозиция

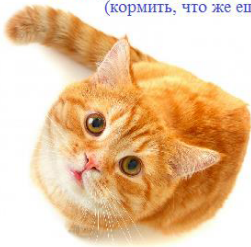
Принципы ООП

Отношения между классами и UML диаграммы

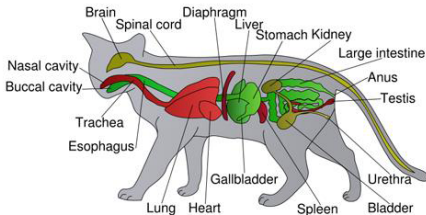
# Принципы ООП

- ▶ **Абстрагирование (Abstraction)** означает выделение значимой информации и исключение из рассмотрения не значимой.

понятно, что делать с объектом  
(кормить, что же еще)

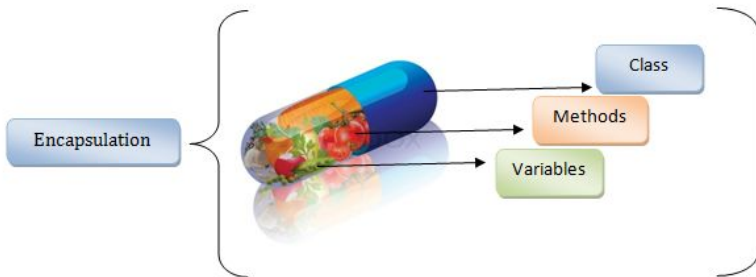


не понятно, как именно взаимодействовать с объектом  
(слишком много деталей)



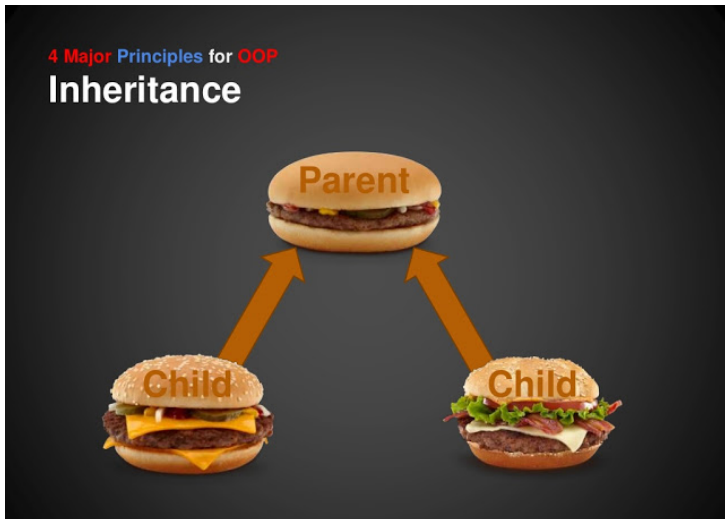
# Принципы ООП

- ▶ **Инкапсуляция (Encapsulation)** - это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных. Доступ к коду и данным жестко контролируется интерфейсом.



# Принципы ООП

- ▶ **Наследование (Inheritance)** касается способности языка позволять строить новые определения классов на основе определений существующих классов.





# Принципы ООП

- **Полиморфизм (Polymorphism)** - свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

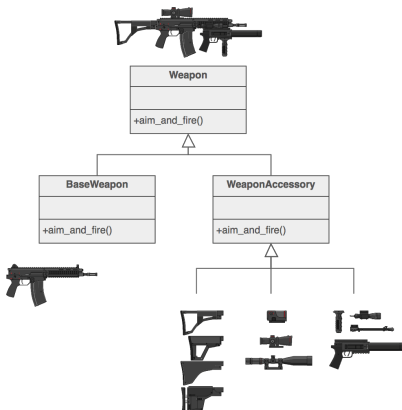


by Sinipull for codecall.net



# Преимущества ООП

- ▶ Использование моделей из окружающего мира - объектов.
- ▶ Объектная декомпозиция.
- ▶ Повторное использование кода (наследование).
- ▶ Скрытие сложности.



# Классы и объекты

**Класс** — это элемент ПО, описывающий абстрактный тип данных и его частичную или полную реализацию.

Класс - универсальный, комплексный тип данных, состоящий из тематически единого набора **полей** (переменных более элементарных типов) и **методов** (функций для работы с этими полями)

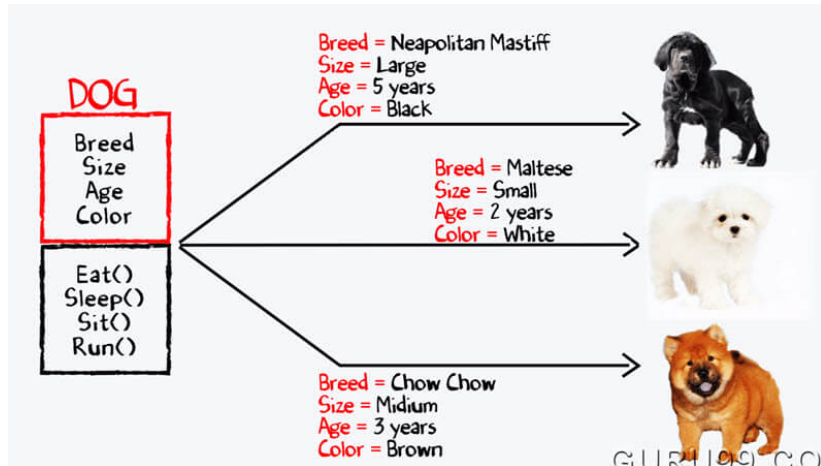
# Классы и объекты

**Объект** - некоторая сущность в компьютерном пространстве, обладающая определённым состоянием и поведением, имеющая заданные значения свойств (атрибутов) и операций над ними (методов).

Объект - это экземпляр класса.

Если в классе может быть определён набор полей (свойств), то в объекте этим полям заданы значения.

# Классы и объекты



Одному классу соответствуют много объектов.

# Основные понятия

**Методы класса** — это его функции.

**Свойства** (атрибуты, поля, информационные члены класса) — его переменные.

**Члены класса** — методы и поля класса.

# Основные понятия

**Интерфейс** совокупность средств, методов и правил взаимодействия (управления, контроля и т.д.) между элементами системы.

**Интерфейс (ООП)** - то, что доступно при использовании класса извне. Как правило это набор методов.

# Основные понятия

Каждый объект характеризуется:

- ▶ Состояние - набор атрибутов, определяющих поведение объекта.
- ▶ Поведение - это то, как объект действует и реагирует; поведение выражается в терминах состояния объекта и передачи сообщений.
- ▶ Идентичность (уникальность) - это такое свойство объекта, которое отличает его от всех других объектов. Два объекта идентичны если представлены одним и тем же участком памяти.
- ▶ Равенство (эквивалентность). Два объекта равны если содержат одинаковые данные.



## Равенство и эквивалентность

```
string *s = new string("ABC");  
string *s1 = new string ("ABC");  
string *s2 = s;
```

s и s1 равны

s и s2 эквивалентны (указывают на один и тот же участок памяти)

# Объявление класса в C++

```
class ClassName {  
private:  
    // закрытые члены класса  
    // рекомендуется для описания полей  
public:  
    // открытые (доступные из вне) члены класса  
    // рекомендуется для описания интерфейса  
protected:  
    // защищенных члены класса  
    // доступны только наследникам  
  
    // дружественные функции и классы  
    // модификатор доступа не важен  
friend заголовок-функции;  
friend имя_класса;  
};
```

## Классы и объекты. Пример. C++

```
class Book {  
    public:  
        string title;  
        string author;  
        unsigned pages;  
};  
  
// объекты (экземпляры класса Book)  
Book b1 = {"Code complete", "S. Macconell", 900};  
Book b2 = {"OOA and OOD", "Grady Booch", 897};  
Book b3 = {"Незнайка на Луне", "Носов. Н", 408};  
  
Book b4 = Book();  
Book b5();  
Book b6;
```

при создании объектов b1, b2 и b3 использованы списки инициализации. Такой способ инициализации подходит для иллюстрации создания класса, но нарушает инкапсуляцию.

Объекты b4, b5, b6 будут равны.

# Классы и объекты. Пример. Python

```
class Book:  
    title = ""  
    author = ""  
    pages = 0
```

```
b1 = Book()  
b1.title = "Code complite"  
b1.author = "S. Macconell"  
b1.pages = 900
```

# Описание полей и методов

- ▶ Поля и методы описываются также как и переменные и функции соответственно.
- ▶ Каждый класс неявно содержит специальный указатель на самого себя - `this`<sup>1</sup>.
- ▶ Этот указатель неявно передаётся первым параметром в каждый метод<sup>2</sup>.
- ▶ Если для члена класса не указан модификатор доступа, то считается что он `private`

---

<sup>1</sup> в python это `self`

<sup>2</sup> в python передаётся явно, например: `__init__(self, a, b)`

## Доступ к членам класса

- ▶ доступ через объект или ссылку на объект - операция выбора члена класса `"."`

```
Book b1;  
b1.author = "Станислав Лем";
```

- ▶ доступ с помощью указателя на объект - указатель на член класса `"->"`

```
Book *b2 = new Book();  
b2->author = "Станислав Лем";  
// аналогично  
(*b2).author = "Станислав Лем";
```

## Пример

```
class X {  
private:  
    int val1;  
public:  
    float val2;  
    int *vec;  
    void foo() {  
        this->val1 = 200;  // this - это указатель  
        val1 = 200;  // this можно не указывать при обращении к членам клас  
    }  
};  
...
```

```
X x;  
X *xp = new X();  
x.val1 = 10;  // Ошибка! Поле val1 недоступно извне класса.  
x.val2 = 10;  
x.vec = NULL;  
x.foo();  
xp->val2 = 9000;  
xp->foo();
```

# Некоторые рекомендации.

## Парадигма ООП

- ▶ Поля класса рекомендуется описывать в закрытой области класса (`private`).
- ▶ Для доступа к таким полям создавать методы для получения и задания значения (инкапсуляция).
- ▶ Эти методы должны быть доступны извне класса (`public`)
- ▶ Константность: методы не изменяющие состояние класса нужно помечать спецификатором `const`

```
void foo() const {  
    ...  
}
```



# Инкапсуляция

- ▶ Если поля класса не доступны извне класса (`private`) ...
- ▶ То для доступа к ним (изменения и получения значения) используются методы
- ▶ Методы, при необходимости, включают в себя предусловия (см. ADT)
- ▶ Таким образом, у пользователя класса не будет возможности изменить данные неправильным способом
- ▶ Такое объединение данных и методов работы с ними называется **инкапсуляцией**
- ▶ Благодаря инкапсуляции с объектом можно работать как с *чёрным ящиком*

# Инкапсуляция

- ▶ Инкапсуляция не обязательно подразумевает *сокрытие данных*
- ▶ Например в Python к данным класса (полям) можно получить непосредственный доступ
- ▶ Тем не менее объект в Python включает в себя методы для работы с данными, а значит принцип инкапсуляции соблюдается

# Некоторые рекомендации

## Стиль кодирования

- ▶ Классы рекомендуется называть используя верблюжью нотацию - CamelCase
- ▶ рекомендуемые имена методов для обращения к полю класса - см. пример
- ▶ Определение класса следует разделять на заголовочный (\*.h) и cpp файл. В cpp файле должны приводится только определения (definition) методов.
- ▶ Имя заголовочного файла должно совпадать с именем класса.

# Пример

## Заголовочный файл MyClass.h

```
class MyClass{
    int _x;

public:
    MyClass();

    int x() const;
    void setX(int x);

    void foo( int x, int y);
};
```

## MyClass.cpp

```
MyClass::MyClass(){
    _x = 42;
    ... }

int MyClass::x() const{
    return _x;}

void MyClass::setX(int x){
    // проверка входных данных
    // если всё ОК:
    _x = x; }

void MyClass::foo(int x, int y){
    ... }
```

# Outline

Абстрактный тип данных

Введение в ООП

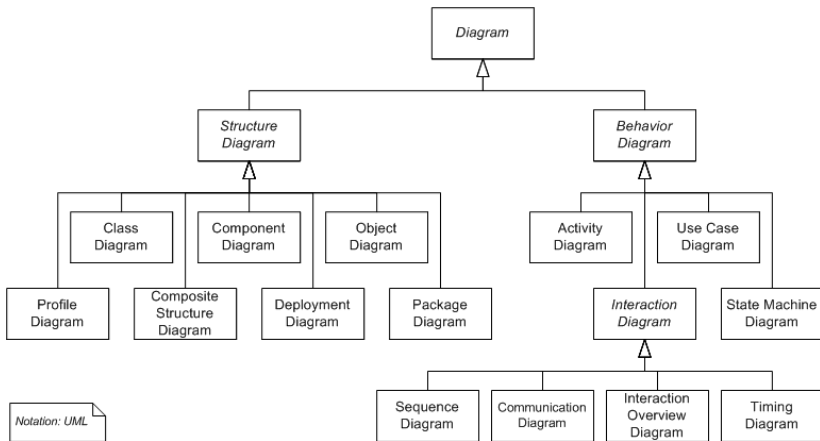
Декомпозиция

Принципы ООП

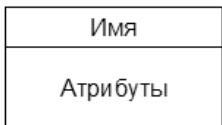
Отношения между классами и UML диаграммы

**UML** (Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

Виды UML диаграмм представленные в виде UML диаграммы.

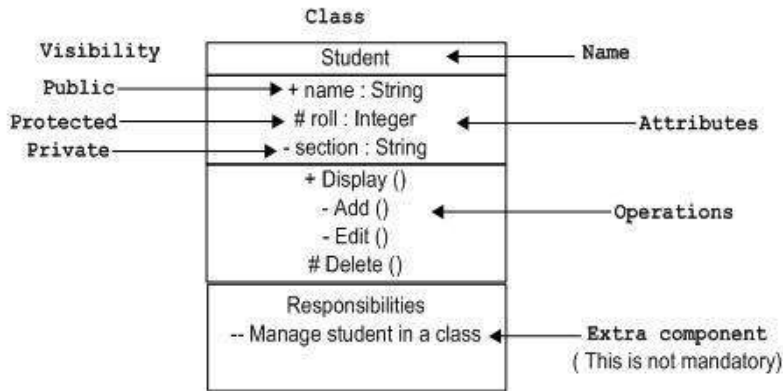


## Класс на UML диаграмме





# Класс на UML диаграмме



# Отношения между классами

- ▶ обобщение/специализация (generalization/specialization)

кошки - это животные

- ▶ целое/часть (whole/part)

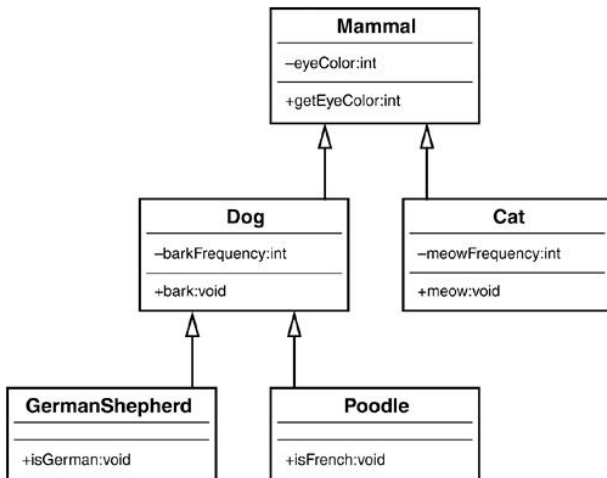
двигатель - часть автомобиля

- ▶ ассоциация (семантическая зависимость)

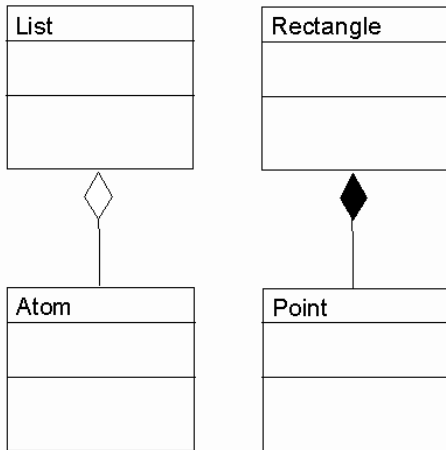
художник - кисть

# Обобщение\специализация

## Наследование



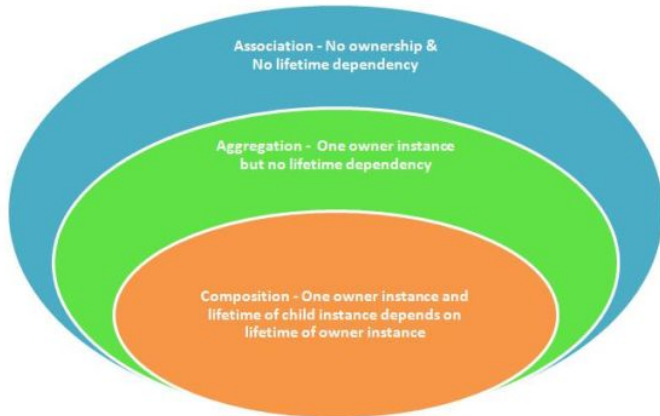
## Агрегация и композиция



# Ассоциация



# Ассоциация - Агрегация - Композиция



## Ссылки и литература

1. 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.
2. MSDN - Microsoft Developer Network
3. 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.
4. [www.stackoverflow.com](http://www.stackoverflow.com) - система вопросов и ответов
5. [draw.io](http://draw.io) — создание диаграмм.

# Материалы курса

Слайды, вопросы к экзамену, задания, примеры

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

