

# Объектно-ориентированное программирование

Введение в ООП

ИВТ и ПМ  
ЗабГУ

2021

# План

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

Вопросы

Ссылки и литература

# Outline

## Введение в ООП

### Декомпозиция

### Абстрактный тип данных

#### Пример

### Класс

#### Абстрагирование

#### UML диаграмма класса

#### Классы и объекты

### Классы в C++

#### Модификаторы доступа, методы и инкапсуляция

### Принципы ООП

### Вопросы

### Ссылки и литература

# Структурное программирование

- ▶ Что такое структурное программирование? [\[wiki\]](#)
- ▶ Что такое процедурное программирование?
- ▶ Что такое модульное программирование?

# Outline

Введение в ООП

**Декомпозиция**

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

Вопросы

Ссылки и литература

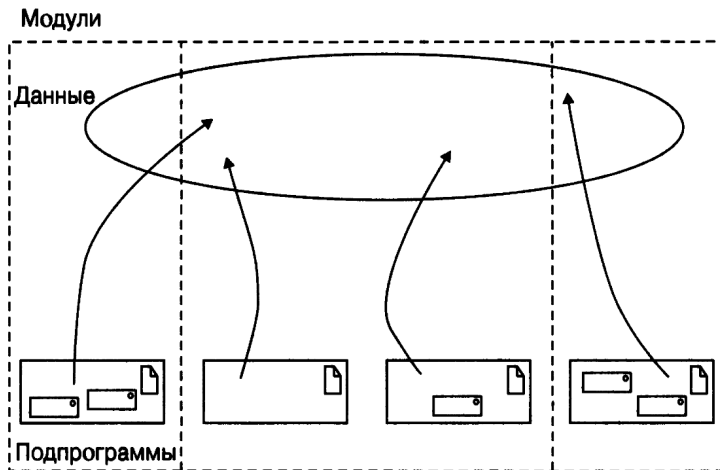
# Декомпозиция

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

# Декомпозиция

- ▶ **Алгоритмическая декомпозиция** используется в структурном программировании.
  - ▶ Задачи разбиваются на подзадачи
  - ▶ Решение задачи и сама программа – *процесс*
  - ▶ Структуры данных – вторичны
- ▶ Код решающий задачи можно вынести в подпрограммы
- ▶ Подпрограммы (и типы данных) можно объединить в модули

# Модульное программирование

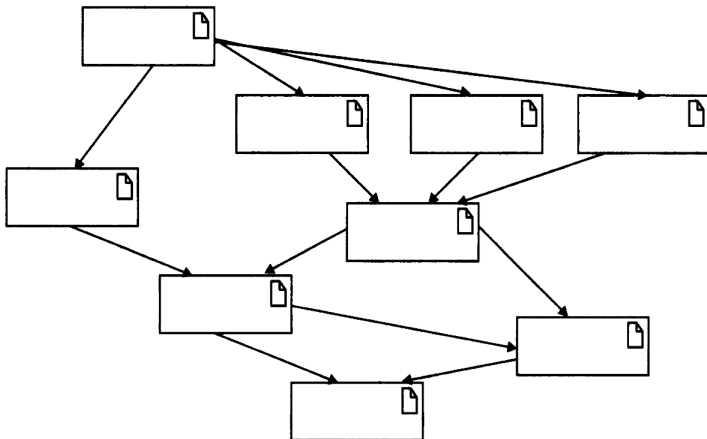




Однако мир представляет собой совокупность взаимодействующих объектов...

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

# Объектно-ориентированное программирование



- ▶ Практически все языки программирования поддерживают ООП
- ▶ ООП – самая популярная парадигма программирования
- ▶ ООП хорошо сочетается с событийно-ориентированным программированием

# Объектно-ориентированная декомпозиция

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

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

# Объектно-ориентированная декомпозиция

## Пример.

- ▶ Главное окно программы можно представить объектом
- ▶ Этот объект состоит из других объектов - кнопок, надписей, полей ввода и т.д.
- ▶ Файл – тоже представляется объектом.
- ▶ В файле может храниться набор сведений (таблица), например телефонная книга.
- ▶ Каждая запись в телефонной книге – объект.
- ▶ Эти объекты можно показать (в том или ином виде) в таблице на главном окне.
- ▶ Объект файл – читает содержимое конкретного файла. Создаёт на основе прочитанной информации набор других объектов – записей телефонной книги.
- ▶ Главное окно записывает загруженные записи в таблицу
- ▶ Таблица отображает данные на главном окне

# Объект

*"**Объект** – это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области"*

– Г. Буч

# Объекты

Примеры объектов?

# Объект

- ▶ Объект обладает поведением – может что-то делать  
Кот умеет мяукать, спать и есть
- ▶ Объект обладает характеристиками – по ним можно отличить один объект от другого  
Этот кот белый, с зелёными глазами и мягкой шёрсткой
- ▶ Объект важен в предметной области



# Пример

## Процедурное программирование

```
struct GeoCoord{
    float lat;           // широта
    float lon;          // долгота
};

// прибавить delta градусов к широте
void add_longitude(GeoCoord &c, float delta){
    c.lat += delta;
    if ( fabs(c.lat) ) > 90 { ... }
}

// ...

GeoCoord c = {51, 110};
// ...
c.lon = 1000;           // свойства объекта нарушаются!

add_lat( c, 50 );
```

# Пример

## Объектно-ориентированное программирование

```
struct GeoCoord{
    private:
        float lat;          // широта
        float lon;          // долгота
        //..
    public:
        void set_longitude(float x){
            if ( fabs(x) > 180) throw invalid_argument("...");
            else this->lon = x;
        }
        // прибавить delta градусов к широте
        void add_longitude(float delta){
            this->lat += delta;
            if ( fabs(this->lat) ) > 90 { ... }
        }
};

// ...
GeoCoord c(51, 110);           // c — объект
// ...
c.set_longitude( 1000 );
c.add_latitude( 50 );
```

# Outline

Введение в ООП

Декомпозиция

**Абстрактный тип данных**

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

Вопросы

Ссылки и литература

# Объекты могут быть похожи

- ▶ Некоторые объекты могут обладать одинаковым набором свойств.
- ▶ Например записи в телефонной книге.
- ▶ У этих объектов два свойства: номер телефона и имя абонента.
- ▶ Другие объекты могут обладать другим набором свойств
- ▶ Поэтому возникает необходимость описывать общие свойства для групп объектов чтобы отличать их друг от друга.

# Объекты могут быть похожи

- ▶ Над объектами с одинаковым набором свойств можно совершать одинаковый набор действий
- ▶ Например над объектами - записями телефонной книги - можно совершать действия:
  - ▶ изменения номера телефона,
  - ▶ изменение имени абонента,
  - ▶ форматирование номера телефона (преобразование +79991234567 -> +7-999-123-45-57)

# Объекты могут быть похожи

Для описания общности объектов с одинаковыми свойствами и действиями используется **абстрактный тип данных**.

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

Но как и в языках программирования, в описании АДТ существуют договорённости по структуре и стилю описания



# 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

- ▶ Данные – набор из общих свойств для описываемой общности объектов
- ▶ Операции – действия которые можно совершать над данными
  - ▶ Вход – необходимые входные данные для совершения операции. Могут отсутствовать.
  - ▶ Предусловия – требования к входным данным, при соблюдении которых операция может быть произведена
  - ▶ Процесс – совершаемые действия
  - ▶ Выход – выходные данные, получаемые после совершения действия. Могут отсутствовать.
  - ▶ Постусловия – требования к данным, которые должны быть соблюдены после выполнения действия.

# Объектно-ориентированная декомпозиция

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

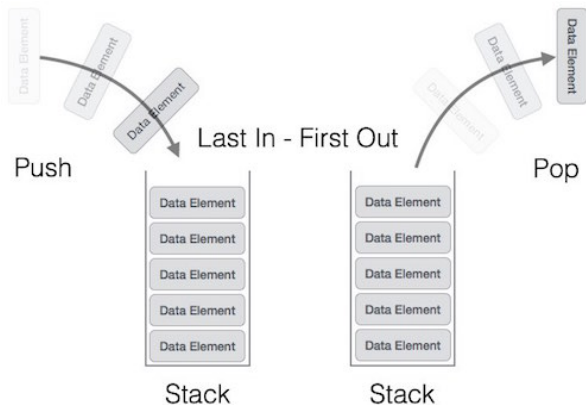
– Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений

- ▶ Изменение данных (значений для свойств) объекта - услуга которую оказывает объект.
- ▶ Чтение кем либо данных из объекта - тоже услуга объекта.
- ▶ Поэтому чтение и изменение данных - это отдельные операции.
- ▶ Такие операции приводят для каждого из свойств объекта.

# 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»)



# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

Вопросы

Ссылки и литература

# 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: Стек 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: Стек имеет новый элемент в вершине

# Пример ADT - Стэк

- ▶ Операции (продолжение)

- ▶ **Peek** Вход: Нет

- Предусловия: Стек не пустой

- Процесс: Нахождение значения элемента в вершине стека

- Выход: Возвращать значение элемента в вершине стека

- Постусловия: Стек неизменный

- ▶ **ClearStack**

- Вход: Нет

- Предусловия: Нет

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

- Выход: Нет

- Постусловия: Стек переустановлен в начальные условия

Конец ADT Stack



## Чем полезен ADT?

- ▶ Инкапсуляция деталей реализации. Это означает, что единожды инкапсулировав детали реализации работы АД мы предоставляем клиенту интерфейс, при помощи которого он может взаимодействовать с АД. Изменив детали реализации, представление клиентов о работе АД не изменится.

## Чем полезен ADT?

- ▶ Инкапсуляция деталей реализации. Это означает, что единожды инкапсулировав детали реализации работы АД мы предоставляем клиенту интерфейс, при помощи которого он может взаимодействовать с АД. Изменив детали реализации, представление клиентов о работе АД не изменится.
- ▶ Снижение сложности. Путем абстрагирования от деталей реализации, мы сосредотачиваемся на интерфейсе, т.е на том, что может делать АД, а не на том как это делается. Более того, АД позволяет нам работать с сущностью реального мира.

## Чем полезен ADT?

- ▶ 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.

## Чем полезен АДТ?

- ▶ 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

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

**Класс**

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

Вопросы

Ссылки и литература

# Классы и объекты

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

# 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

**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)

**Класс** – это множество объектов, обладающих общей структурой, поведением и семантикой.



# Классы

- ▶ Какие поля должны быть у классов представляющих ... ?
  - ▶ Студента
  - ▶ Программиста
  - ▶ Шоколадный батончик
  - ▶ Легковой автомобиль
  - ▶ Курсовую работу
  - ▶ Вектор на плоскости
  - ▶ Электронное письмо
  - ▶ Запись трудовой книжке
- ▶ Какие методы должен иметь каждый из классов?

# Классы

- ▶ Набор полей и методов в классе зависит ещё и от предметной области, в которой класс будет рассматриваться.
- ▶ Для владельца автомобиля важны одни его характеристики, а для автомеханика - другие. Хотя частично эти характеристики для владельца и автомеханика совпадают.
- ▶ *Приведите примеры полей, которые могут быть в классе, с точки зрения автосервиса и с точки зрения автовладельца (который не занимается самостоятельным ремонтом).*

# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

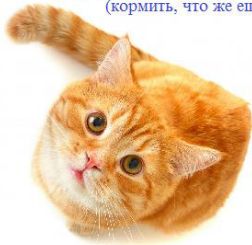
Вопросы

Ссылки и литература

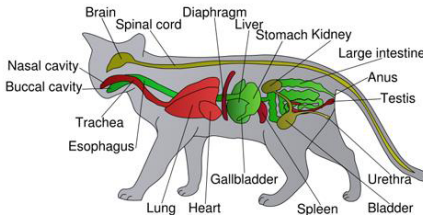
# Принципы ООП

- ▶ **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)



Класс не должен содержать лишней информации (полей)

# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

**Класс**

Абстрагирование

**UML диаграмма класса**

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

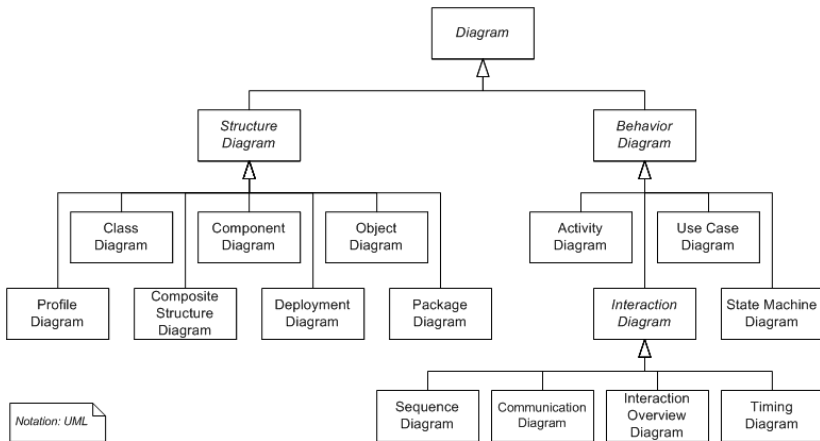
Вопросы

Ссылки и литература

- ▶ Проектирование классов должно происходить до их кодирования
- ▶ Для описания классов и отношений между ними используется графический язык - UML

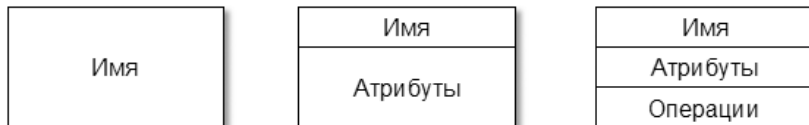
**UML** (Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

Виды UML диаграмм представленные в виде UML диаграммы.





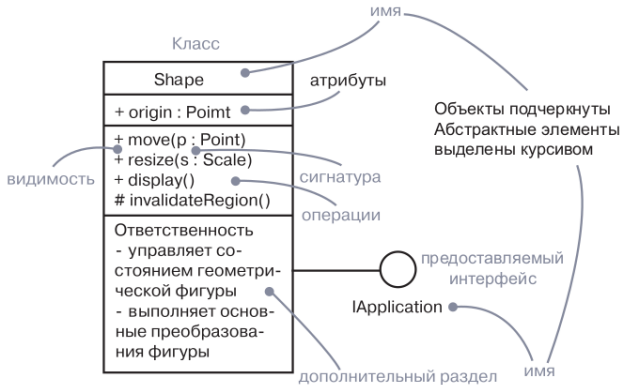
# Класс на UML диаграмме



Три способа изображения класса, с разным уровнем детализации его членов.

# Класс на UML диаграмме

## Пример



+ public  
# protected  
- private

# Класс на UML диаграмме

## Инструменты для создания диаграммы классов

- ▶ **Qt Creator**  
в существующем проекте: добавить новый – моделирование – модель
- ▶ **Microsoft Visio**
- ▶ сайт **draw.io**  
выбрать: software - class
- ▶ **Umbrello**  
Генерация кода из UML диаграммы
- ▶ **Modellio**  
Продвинутый инструмент. Относительно сложный интерфейс
- ▶ **plantuml.com** – создание диаграмм в plain text

# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

**Класс**

Абстрагирование

UML диаграмма класса

**Классы и объекты**

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

Вопросы

Ссылки и литература

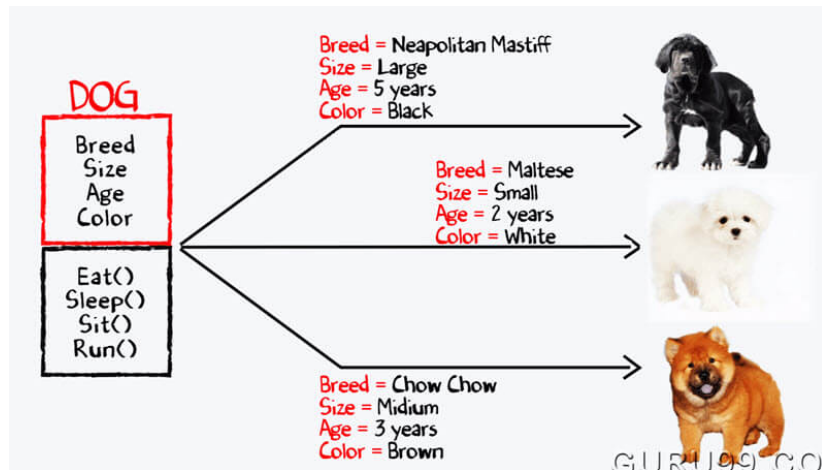
# Классы и объекты

**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



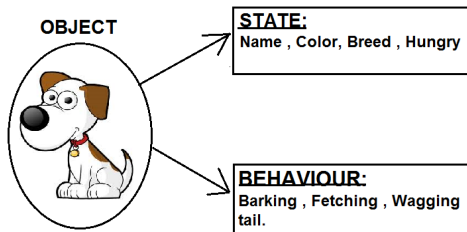
Одному классу Dog соответствуют много объектов.

# Основные понятия

Каждый объект характеризуется:

- ▶ **Состояние** (state) – набор атрибутов, определяющих поведение объекта.
- ▶ **Поведение** (behaviour) – это то, как объект действует и реагирует; поведение выражается в терминах состояния объекта и передачи сообщений.

# Состояние и поведение



state – состояние  
behaviour – поведение



# Основные понятия

Каждый объект характеризуется:

- ▶ **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)

# 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.

# Outline

- Введение в ООП

- Декомпозиция

- Абстрактный тип данных

  - Пример

- Класс

  - Абстрагирование

  - UML диаграмма класса

  - Классы и объекты

- Классы в C++**

  - Модификаторы доступа, методы и инкапсуляция

- Принципы ООП

- Вопросы

- Ссылки и литература

# Классы

Класс – это ...

- ▶ ... элемент ПО, описывающий абстрактный тип данных и его частичную или полную реализацию.
- ▶ ... тип данных, который создаёт программист.

# Классы

Класс – это ...

- ▶ ... элемент ПО, описывающий абстрактный тип данных и его частичную или полную реализацию.
- ▶ ... тип данных, который создаёт программист.
- ▶ *Как назывались типы данных, которые вы создавали в других языках программирования?*

# Объявление класса в C++

Общая структура

```
class ClassName {  
  private:  
    // закрытые члены класса  
    // доступны только классу  
    // рекомендуется для описания полей  
  public:  
    // открытые (доступные извне) члены класса  
    // рекомендуется для описания интерфейса  
    // (методы: геттеры, сеттеры, ... )  
  protected:  
    // защищенные члены класса  
    // доступны только наследникам  
  
  // дружественные функции и классы  
  // модификатор доступа не важен  
  friend заголовок-функции;  
  friend имя_класса;  
};
```



# Поля класса.

Пример на C++: объявление класса с полями

```
class Book {  
    public:  
        // открытая область класса  
        // Поля класса:  
        string title;  
        string author;  
        unsigned pages;  
};
```

Этот класс состоит только из открытых полей и является аналогом структуры (struct) в Си или записи (record) в Паскале.

# Поля класса

Пример на C++: Создание экземпляра класса

**Абстрагирование.** Опишем класс для книг в самом простом варианте, оставив только название, автора и количество страниц.

```
// объекты (экземпляры класса Book)
// Такие способы создания объектов эквивалентны
Book b6;
Book b4 = Book();

// Создание объекта и инициализация полей
Book b1 = {"Code complete", "S. Macconell", 900};
Book b2 = {"OOA and OOD", "Grady Booch", 897};
Book b3 = {"Незнайка на Луне", "Носов. Н", 408};
```

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

Объекты b4, b6 будут равны.

## Доступ к членам класса

- ▶ доступ через объект или ссылку (reference) на объект

```
Book b1; // объект
```

```
b1.author = "Станислав Лем";
```

```
Book &b2 = b1; // ссылка на объект
```

```
b2.pages = 129;
```

- ▶ Оператор точка – оператор доступа к членам класса

## Доступ к членам класса

- ▶ Однако, часто объекты нужно создавать динамически, во время работы программы.

```
// Динамическое создание объекта  
Book *b2 = new Book();
```

- ▶ Для обращения к таким объектам используется указатель (pointer)
- ▶ Чтобы обратиться к членам такого класса, например полям приходится *разыменовывать* указатель:

```
// Динамическое создание объекта  
(*b2).author = "Станислав Лем";
```

- ▶ Такая запись неудобна...

## Доступ к членам класса

- ▶ Поэтому с C++ есть оператор доступа с помощью указателя на объект "->"

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

- ▶ Для этого оператора не нужно делать разыменование указателя

# Классы и объекты. Пример. Python

```
class Book:
```

```
    title = ""
```

```
    author = ""
```

```
    pages = 0
```

```
b1 = Book()
```

```
b1.title = "Code complite"
```

```
b1.author = "S. Macconell"
```

```
b1.pages = 900
```

# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

**Классы в C++**

**Модификаторы доступа, методы и инкапсуляция**

Принципы ООП

Вопросы

Ссылки и литература

# Контролируй данные

- ▶ Рассмотрим класс представляющий время (24-часовой формат)

```
class Time{  
public:  
    uint8_t hours;  
    uint8_t minutes;
```



# Контролируй данные

- ▶ Рассмотрим класс представляющий время (24-часовой формат)

```
class Time{  
public:  
    uint8_t hours;  
    uint8_t minutes;
```

- ▶ Проблема такого класса в том, что контроль за содержанием полей `_hours` и `_minutes` ложится на программиста, который им пользуется.
- ▶ Программист должен каждый раз проверять, не записал ли он туда случайно недопустимые значения

```
Time t;  
t.minutes = 200;
```

- ▶ Программисту придётся контролировать *предусловия* самостоятельно.

# Контролируй данные

- ▶ Решение: запретить прямой доступ к полям
- ▶ Но чтобы возможность изменять данные осталась, нужно реализовать *методы* – специальные функции внутри класса.
- ▶ Эти методы будут контролировать предусловия и постусловия
- ▶ Таким образом методы – это операции АДТ реализованные на языке программирования

# Контролируй данные

Для контроля доступа к членам класса используются *модификаторы доступа*.

- ▶ `public` (открытый) – члены класса доступны всем
- ▶ `private` (закрытый) – члены класса доступны только изнутри класса
- ▶ `protected` (закрытый) – члены класса доступны только изнутри класса и наследникам

# Контролируй данные

- ▶ В C++ возможно объявить поля класса сделав их недоступными извне класса (область `private` и `protected`) - *принцип сокрытия*
- ▶ Для доступа к полям тогда нужно будет создать методы, объявив их в открытой (`public`) области класса
- ▶ Как правило для одного поля приходится создавать два метода:
  - ▶ метод чтения – *getter* (getter) – для получения значения поля класса
  - ▶ модифицирующий метод – *setter* (setter) – для задания значения полю класса. сеттер как раз и включает в себя проверку предусловий

# Контролируй данные

## Пример

```
// класс для хранения времени в 24-часовом формате
class Time{
private: // закрытая часть класса (недоступна извне класса)
    uint8_t _hours;
    uint8_t _minutes;

public: // открытая часть класса (доступна всем)
    Time(){_hours=0; _minutes=0;}

    uint8_t hours() {return _hours;} // геттер: возвращает часы
    uint8_t minutes() {return _minutes;} // геттер: возвращает минуты

    void setHours(uint8_t h){ // сеттер: задаёт часы
        if (h >= 0 && h < 24) // проверка предусловий
            _hours = h;
    }

    void setMinutes(uint8_t m){ // сеттер: задаёт минуты
        if (m >= 0 && m < 60) // проверка предусловий
            _minutes = m;
    }
};
```

# Инкапсуляция

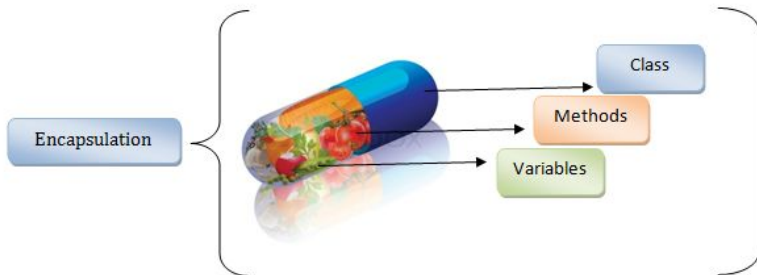
- ▶ Если поля класса не доступны извне класса (private) ...
- ▶ То для доступа к ним (изменения и получения значения) используются методы
- ▶ Методы, при необходимости, включают в себя предусловия (см. ADT)
- ▶ Таким образом, у программиста не будет возможности изменить данные неправильным способом пользуясь классом
- ▶ 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

# 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.





# Описание полей и методов

- ▶ Каждый класс неявно содержит специальный указатель на самого себя – `this`<sup>1</sup>.

```
void setHours(uint8_t h){  
    if (h >= 0 && h < 24)  
        _hours = h;    }
```

// аналогично

```
void setHours(uint8_t h){  
    if (h >= 0 && h < 24)  
        this->_hours = h;    }
```

- ▶ Этот указатель неявно передаётся первым параметром в каждый метод<sup>2</sup>.

---

<sup>1</sup> в python это `self`

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

# Некоторые рекомендации.

## Парадигма ООП

- ▶ Поля класса рекомендуется описывать в закрытой области класса (`private`).
- ▶ Если для члена класса не указан модификатор доступа, то считается что он `private`
- ▶ Для доступа к таким полям создавать методы для получения и задания значения (инкапсуляция).
- ▶ Эти методы должны быть доступны извне класса (`public`)

# 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 приводиться 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

```
#include "MyClass.h"

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

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

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

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

Оператор :: используется неслучайно, ведь класс это частный случай пространства имён.

## Замечание о способах обмена информации

- ▶ Взаимодействие класса с внешними объектами должно быть реализовано исключительно с помощью методов (и дружественных функций)
- ▶ Если класс должен получить некоторые данные, то необходимо определить метод, который примет эти данные через свои параметры

```
class MyClass{  
    float x;  
    public:  
        void set_x(float x1) {  
            // ... обработка условий...  
            this->x = x1;        }  
};  
MyClass my_object;  
float a;  
cin >> a;  
my_object.set_x(a);
```

- ▶ Использование cin или других способов получения данных критикуется так же как функции с побочными эффектами.

# Замечание о способах обмена информации

```
// Посылка
class Package{
    float weight;          // вес
    // todo: другие поля
public:

    // плохо: ввод возможен только из консоли
    void set_weight() {
        cout << "Введите вес" << endl;
        cin >> weight; }

    // хорошо: метод не зависит от способа получения значения w
    void set_weight(float w) {
        // проверка предусловия: не доверяем входным данным
        if (w > 0)
            this->weight = w;
        else{}
    };
};

Package p;
float w = ... // получаем вес любым способом
p.set_weighth(w);
```

## Замечание о способах обмена информации

- ▶ То же касается и вывода данных (например с помощью cout) внутри методов (если только класс специально для этого не создан)

```
class MyClass{  
    float x;  
    public:  
    // ...  
    float get_x() {  
        return this->x;}  
};  
MyClass my_object;  
// ...  
float a = my_object.get_x();
```

- ▶ Тем не менее в этом курсе будут встречаться примеры нарушающие эти правила, однако сделано это будет только для иллюстрации рассматриваемых механизмов и концепций ООП

# Замечание о способах обмена информации

```
// Посылка
class Package{
    float weight;          // вес
    // todo: другие поля
public:

    // плохо: вывод только в консоль; нельзя записать в переменную
    void get_weight_bad() {
        cout << "Bec" << weight << endl;}

    // хорошо: метод не зависит от способа получения значения w
    float get_weight() {
        return this->w; }
};

Package p;
// ...
float w = p.get_weight();
// делаем с переменной w всё что угодно: выводим в консоль,
// записываем в файл, показываем в окне
```



# Повышаем удобство использования класса

```
// Посылка
class Package{
    float weight;           // вес
    float height, width, length; // высота, ширина, длина
    // todo: другие поля
public:

    void set_weight(float w);
    void set_height(float h);
    void set_width(float w);
    void set_length(float l);
    // todo: другие методы
};
```

```
Package p;
// плохо: для инициализации нужно 4 метода
p.set_weight(13.0);
p.set_height(0.4);
p.set_widht(0.55);
p.set_length(0.7);
```

# Повышаем удобство использования класса

```
// Посылка
class Package{
    float weight;           // вес
    float height, width, length; // высота, ширина, длина
    // todo: другие поля
public:

    // конструктор с параметрами
    Package(float weight1, height1, width1, length1) {
        // хорошо: вызов методов, которые проверяют предусловия
        set_weight(weight1);
        set_height(height1);
        set_width(width1);
        set_length(length1);
    }

    void set_weight(float w);
    void set_height(float h);
    void set_width(float w);
    void set_length(float l);
    // todo: другие методы
};

// хорошо: инициализация объекта в одну строку
Package p(13.0, 0.4, 0.55, 0.7);
```

# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

**Принципы ООП**

Вопросы

Ссылки и литература

# Принципы ООП

Чтобы программировать согласно концепции ООП недостаточно просто писать код с классами. Код должен быть хорошо организован и иметь эффективную архитектуру.

# Принципы ООП

Чтобы программировать согласно концепции ООП недостаточно просто писать код с классами. Код должен быть хорошо организован и иметь эффективную архитектуру.

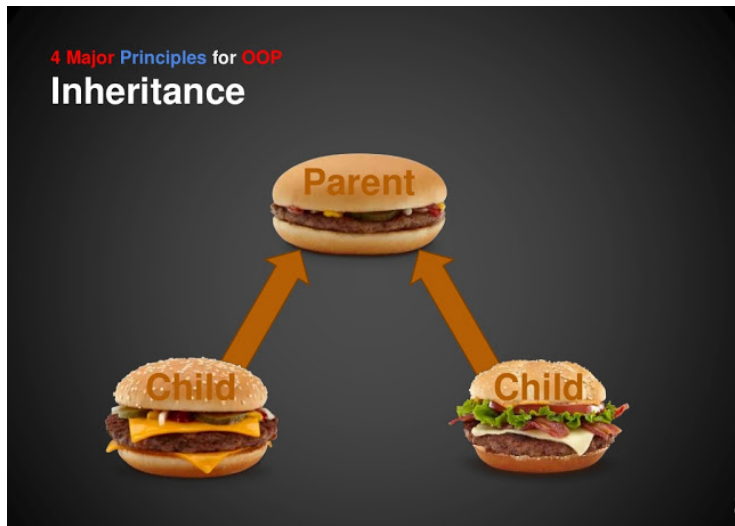
Для этого должны использоваться 4 принципа:

- ▶ Абстрагирование
- ▶ Инкапсуляция
- ▶ Наследование
- ▶ Полиморфизм

Наследование и полиморфизм будут рассмотрены в следующих лекциях

# Принципы ООП

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



# Принципы ООП

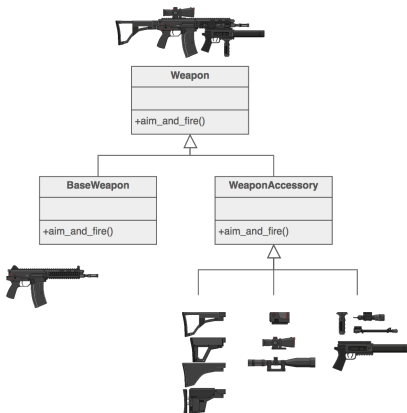
- **Полиморфизм (Polymorphism)** - свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.



by Sinipuli for codecall.net

# Преимущества ООП

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





# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

**Вопросы**

Ссылки и литература

# Вопросы

- ▶ Что такое парадигма программирования?
- ▶ Перечислите и охарактеризуйте известные вам парадигмы программирования
- ▶ Что такое стиль кодирования (стандарт оформления кода, code style)? Чем он отличается от парадигмы программирования?
- ▶ Что такое класс? Объект?
- ▶ Как описать класс без языка программирования?
- ▶ Как соотносятся класс и объект?
- ▶ Что такое абстрагирование?
- ▶ Что такое инкапсуляция? Чёрный ящик? Принцип сокрытия?
- ▶ Как принцип сокрытия реализуется на C++?

# Вопросы

- ▶ Опишите рекомендованный стандарт оформления кода

# Outline

Введение в ООП

Декомпозиция

Абстрактный тип данных

Пример

Класс

Абстрагирование

UML диаграмма класса

Классы и объекты

Классы в C++

Модификаторы доступа, методы и инкапсуляция

Принципы ООП

Вопросы

Ссылки и литература

## Ссылки и литература

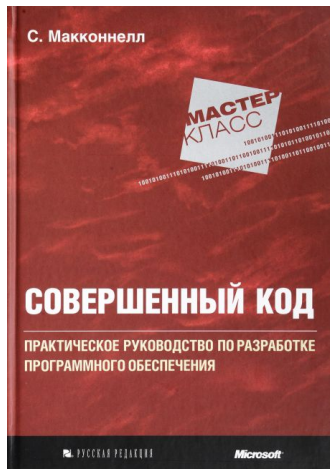


Программирование: принципы и практика с использованием C++, 2-е изд., 2016 (и более поздние издания). - 1328 с.

# Ссылки и литература

1. Курс: Программирование на языке C++  
<https://stepik.org/course/7>
2. 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.
3. MSDN – Microsoft Developer Network
4. [stackoverflow.com](https://stackoverflow.com) – система вопросов и ответов

# Ссылки и литература



Совершенный код. Стив Макконнелл

# Материалы курса

Слайды, вопросы к экзамену, задания, примеры

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

