



Структуры и классы в C++

Антон Кухтичев






Не забудьте
отметиться на
портале!!!

Иначе всё плохо будет.



Содержание занятия

1. Модификаторы доступа
 2. RAI (Resource Acquire Is Initialization)
 3. Константные методы
 4. Наследование
 5. Перегрузка методов
 6. Виртуальные функции
 7. Операторы
- 

Понятие класса

- Объектно-ориентированное программирование построено на понятие класса;
- Объявление класса начинается с ключевого слова `class`;
- По умолчанию члены класса являются закрытыми (`private`-членами);
- Классы и структуры это родственные типы;
- Объект - сущность в адресном пространстве компьютера, появляющаяся при создании класса;
- По определению структура есть класс, все члены которого по умолчанию являются открытыми;

Конструктор

- Конструктор - это функция, которая вызывается при создании объекта;
- Конструктор вызывается автоматически при создании объекта при помощи `new` (но не при помощи `malloc`!);
- Если конструктор не написан явно, C++ гарантирует, что будет создан конструктор по умолчанию;
- Не возвращает тип;



Деструктор

- Деструктор - это функция, которая вызывается при разрушении объекта;
- Если деструктор не написан явно, C++ гарантирует, что будет создан деструктор по умолчанию;
- Не возвращает тип;

Специальные функции-члены

- В C++98 включает четыре такие функции:
 - конструктор по умолчанию
 - деструктор
 - копирующий конструктор
 - оператор копирующего присваивания
- Эти функции создаются, только если они необходимы, т.е. если некоторый код использует их без их явного объявления в классе;
- Конструктор по умолчанию генерируется только в том случае, если в классе не объявлен ни один конструктор.



1. Скотт Мейерс. Эффективное использование C++. Правило 5. Какие функции C++ создаёт и вызывает молча.

Специальные функции-члены

- В C++11 приняты два новых игрока:
 - перемещающий конструктор;
 - оператор перемещающего присваивания;
- Рекомендация о большой тройке
 - Если вы объявили хотя бы одну из трёх операций, то вы должны объявить все три операции.



Ссылка на себя

- Каждая (нестатическая) функция-член занет, для какого объекта она вызвана, и может явно на него ссылаться при помощи `this`;
- `this` является указателем на объект, для которого вызвана функция;

```
struct A
{
    int x_ = 0;
    void foo([A *this]) { this->x_ += 10; }
};
```

RAII (Resource Acquire Is Initialization)


- Захват ресурса есть инициализация.
- В конструкторе объект получает доступ к какому либо ресурсу (например, открывается файл), а при вызове деструктура этот ресурс освобождается (закрывается файл).
- Можно использовать не только для управления ресурсами;
- Класс инкапсулирует владение (захват и освобождение) некоторого ресурса;

Модификатора доступа

```
class A
{
public:
    int x_; // доступно всем;
protected:
    int y_; // доступно только внутри класса и наследникам;
private:
    int z_; // доступно только внутри класса;
};
```




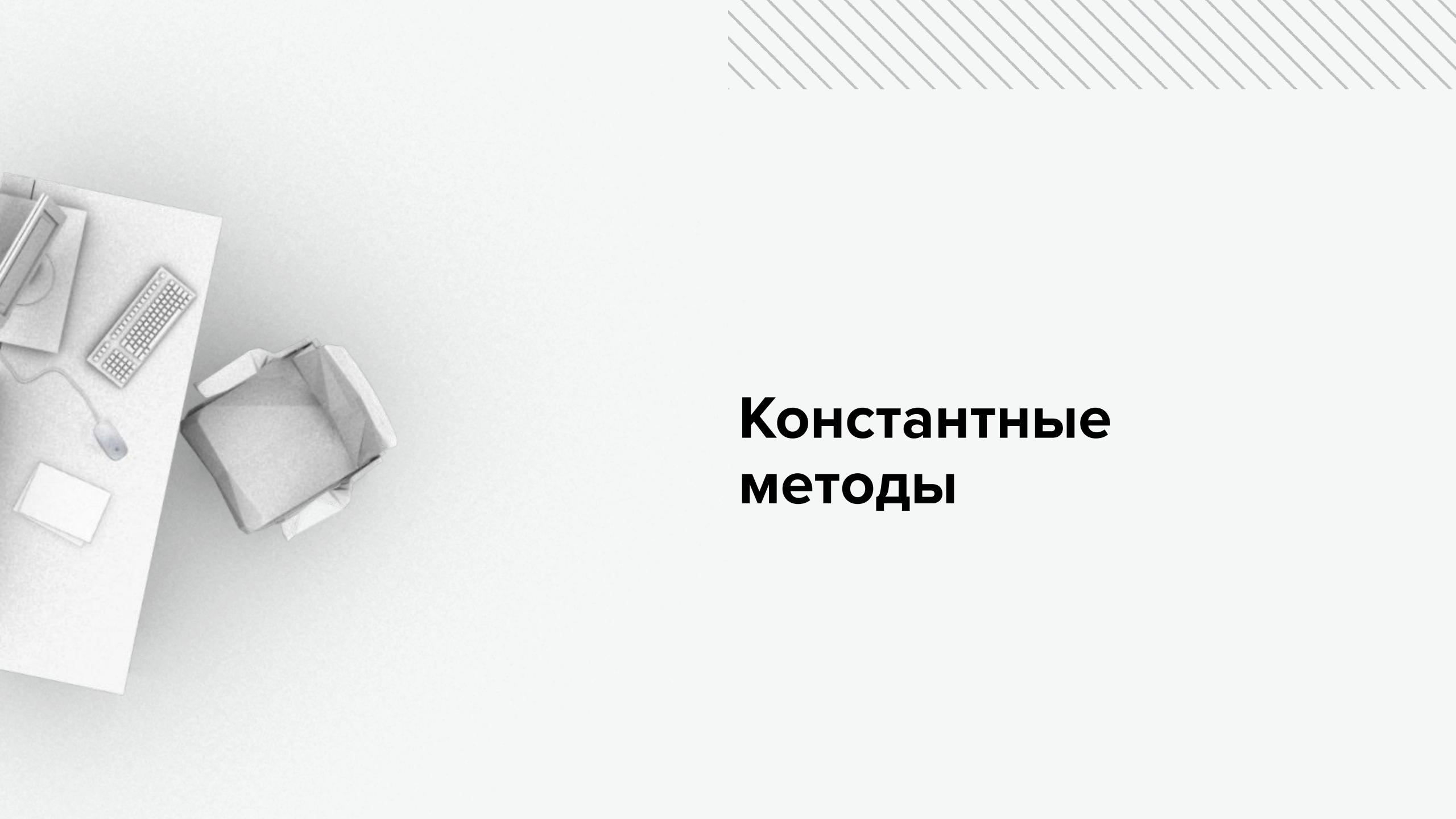
Конструирование объекта

- 
1. Выделяется память под объект;
 2. Если есть базовые классы, то конструирование начинается с них в порядке их очередности в списке наследования;
 3. Инициализируются поля класса в том порядке, в котором они объявлены в классе;
 4. Происходит вызов конструктора.



Уничтожение объекта

- 
1. Происходит вызов деструктора;
 2. Вызываются деструкторы для полей класса в обратном порядке их объявления в классе;
 3. Уничтожаются базовые классы в порядке обратном списку наследования.



Константные методы



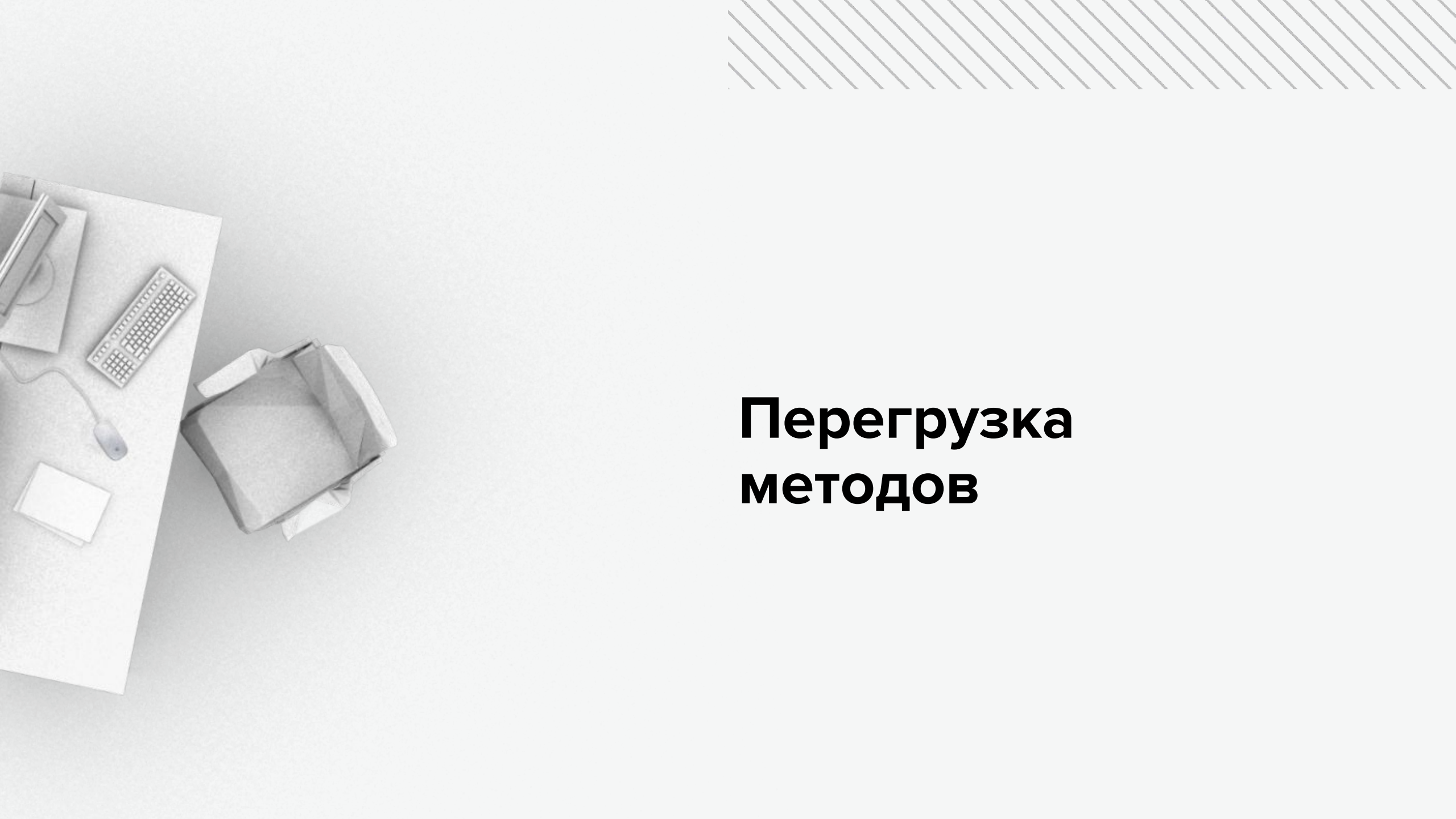
Константные методы

- Любые методы кроме конструктора и деструктора могут быть константными.
- Метод, который гарантирует, что не будет изменять объект или вызывать неконстантные методы класса (поскольку они могут изменить объект).
- Константный метод можно вызывать как для константного, так и для неконстантного объекта, в то время как неконстантный метод можно вызвать только для объекта, не являющегося константой;
- Делайте все ваши методы, которые не изменяют данные объекта класса, константными.



mutable

- Любые методы кроме конструктора и деструктора могут быть константными.



Перегрузка методов



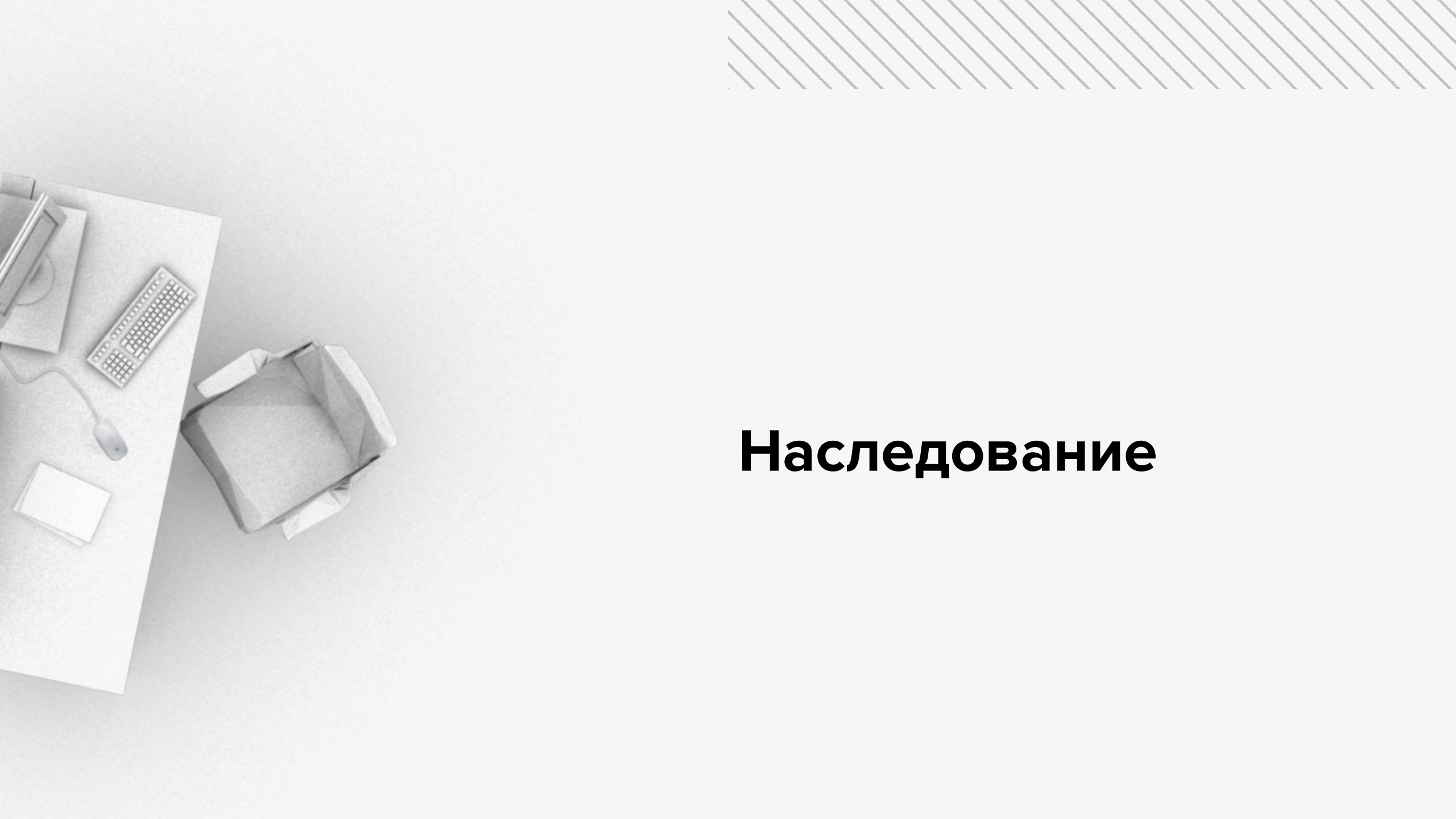
Перегрузка методов

- Методы классов - это просто функции, в которые неявно передается указатель на сам класс;
- Конструкторы - это тоже функции и их тоже можно перегружать.
- Деструкторы - тоже функции, но перегружать нельзя.



Параметры по умолчанию

- Пропусков в параметрах по умолчанию быть не должно, начинаться они могут не с первого аргумента, но заканчиваться должны на последнем.



Наследование



Наследование

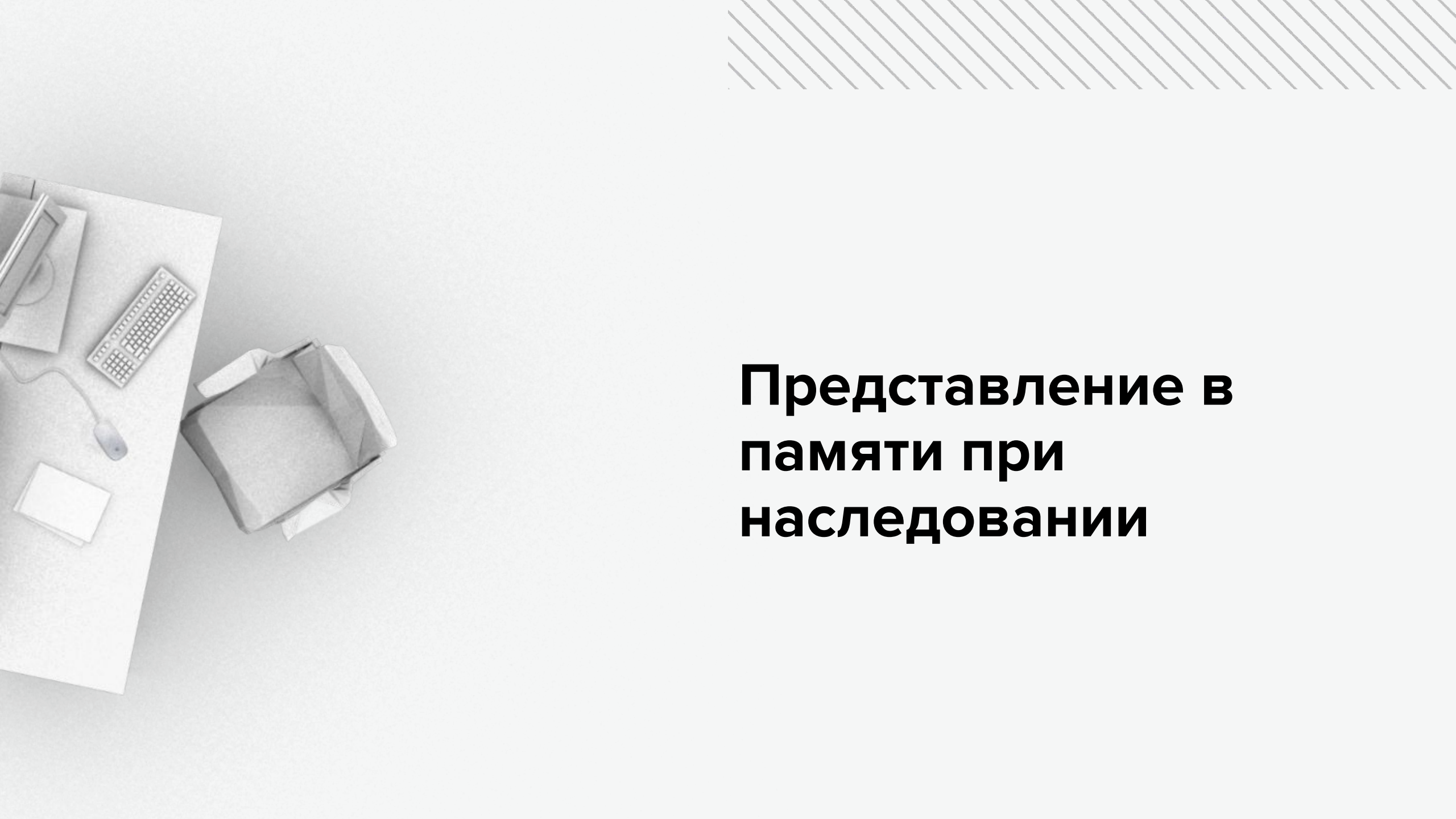
- Возможность порождать класс на основе другого с сохранением всех свойств класса-предка.
- Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником, дочерним или производным классом.
- Наследование моделирует отношение «является».
- Требуется для создания иерархичности – свойства реального мира.

Приведение вверх и вниз по иерархии

- Приведение вверх (к базовому классу) всегда безопасно;
- Приведение вниз может быть опасным;

```
struct A {};  
struct B : public A {};  
struct C : public A {};
```

```
B* b = new B();  
A* a = b;  
C* c = a; // Ошибка компиляции  
C* c = static_cast<C*>(b); // Ошибка компиляции  
C* c = static_cast<C*>(a); // !!!
```



Представление в памяти при наследовании

Инструменты для исследования

- В целях повышения быстродействия данные в памяти должны быть выровнены, то есть размещены определенным образом;
- Предпочтительное выравнивание можно узнать:

```
std::cout << alignof(char) << std::endl; // 1
```

```
std::cout << alignof(double) << std::endl; // 8
```
- `sizeof(T)` - размер типа в байтах
- `offsetof(T, M)` - смещение поля M от начала типа T

Инструменты для исследования

```
struct S
{
    char m1;
    double m2;
};
```

```
sizeof(char) == 1
sizeof(double) == 8
sizeof(S) == 16
offsetof(S, m1) == 0
offsetof(S, m2) == 8
```

```
[          char          ][          double          ]
[c][.][.][.][.][.][.][.][d][d][d][d][d][d][d][d]
```

Простые типы (POD, Plain old data)

1. Скалярные типы (bool, числа, указатели, перечисления (enum), nullptr_t)
2. class или struct которые:
 - a. Имеют только тривиальные (сгенерированные компилятором) конструктор, деструктор, конструктор копирования;
 - b. Нет виртуальных функций и базового класса;
 - c. Все нестатические поля с модификатором доступа public;
 - d. Не содержит статических полей не POD типа.

Простые типы (POD, Plain old data)

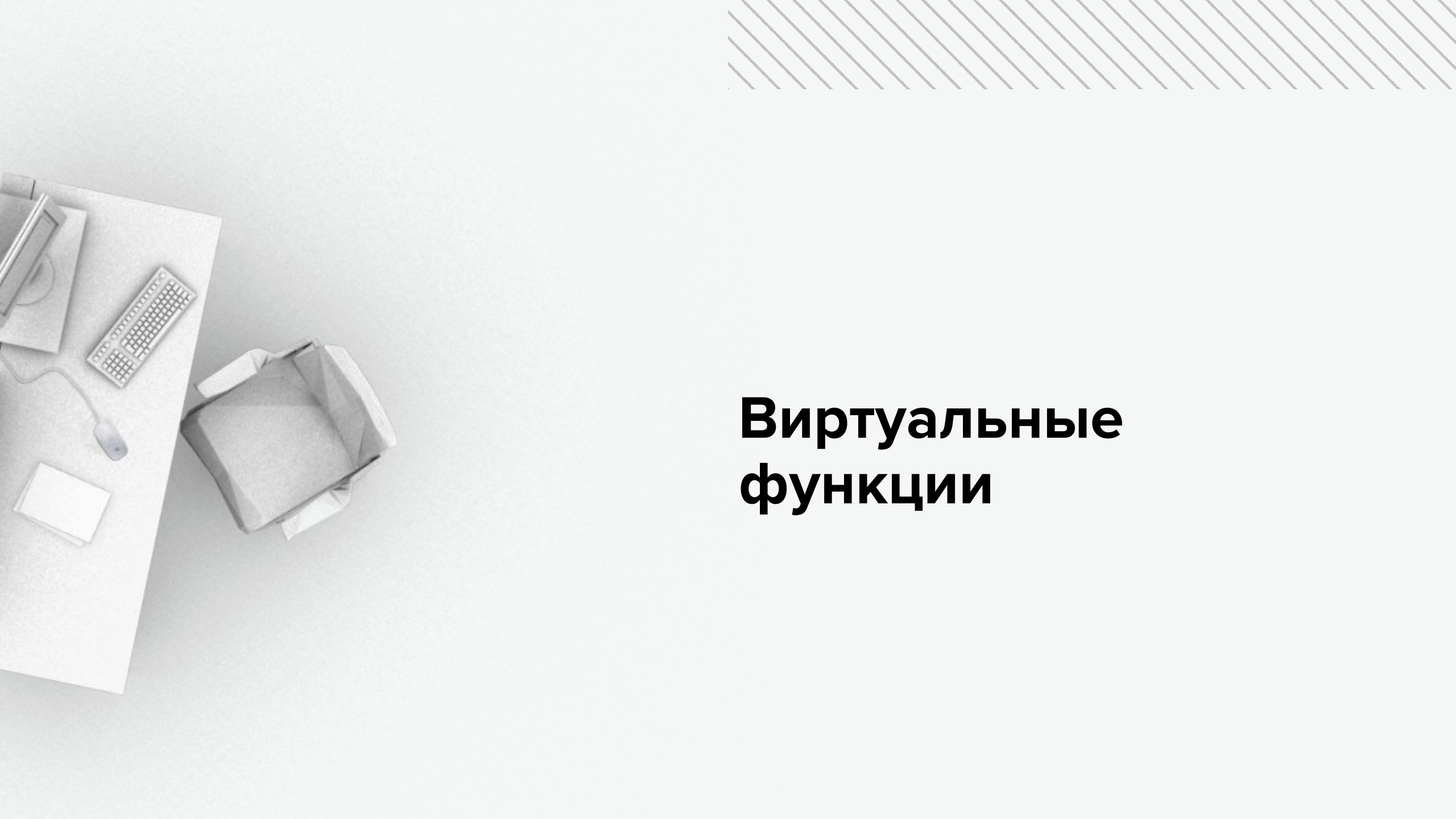
```
class NotPOD
{
public:
    NotPOD(int x)
    {
    }
};
```

```
class NotPOD
    : public Base
{
};
```

```
class NotPOD
{
    virtual void f()
    {
    }
};
class NotPOD
{
    int x;
};
```

Простые типы (POD, Plain old data)

```
class POD
{
public:
    NotPOD m1;
    int m2;
    static double m3;
private:
    void f() {}
};
```



Виртуальные функции

Виртуальные функции

- Решают проблему, связанную с полем типа, предоставляя возможность программисту объявить в базовом классе функции, которые можно заместить в каждом производном классе.
- Производный класс, которые не нуждается в собственной версии виртуальной функции, не обязан её реализовывать;
- Функция из производного класса с тем же именем и с тем же набором типов аргументов, что и виртуальная функция в базовом классе, *замещает* (*override*) виртуальную функцию из базового класса;
- Тип, имеющий виртуальные функции, называется полиморфным типом.

Таблица виртуальных функций

- Если какая-либо функция класса объявлена как виртуальная, создается `vtable`, которая хранит адреса виртуальных функций этого класса;
- Для всех таких классов компилятор добавляет скрытую переменную `vptr`, которая указывает на `vtable`;
- Если виртуальная функция не переопределена в производном классе, `vtable` производного класса хранит адрес функции в родительском классе;

Виртуальные деструктор

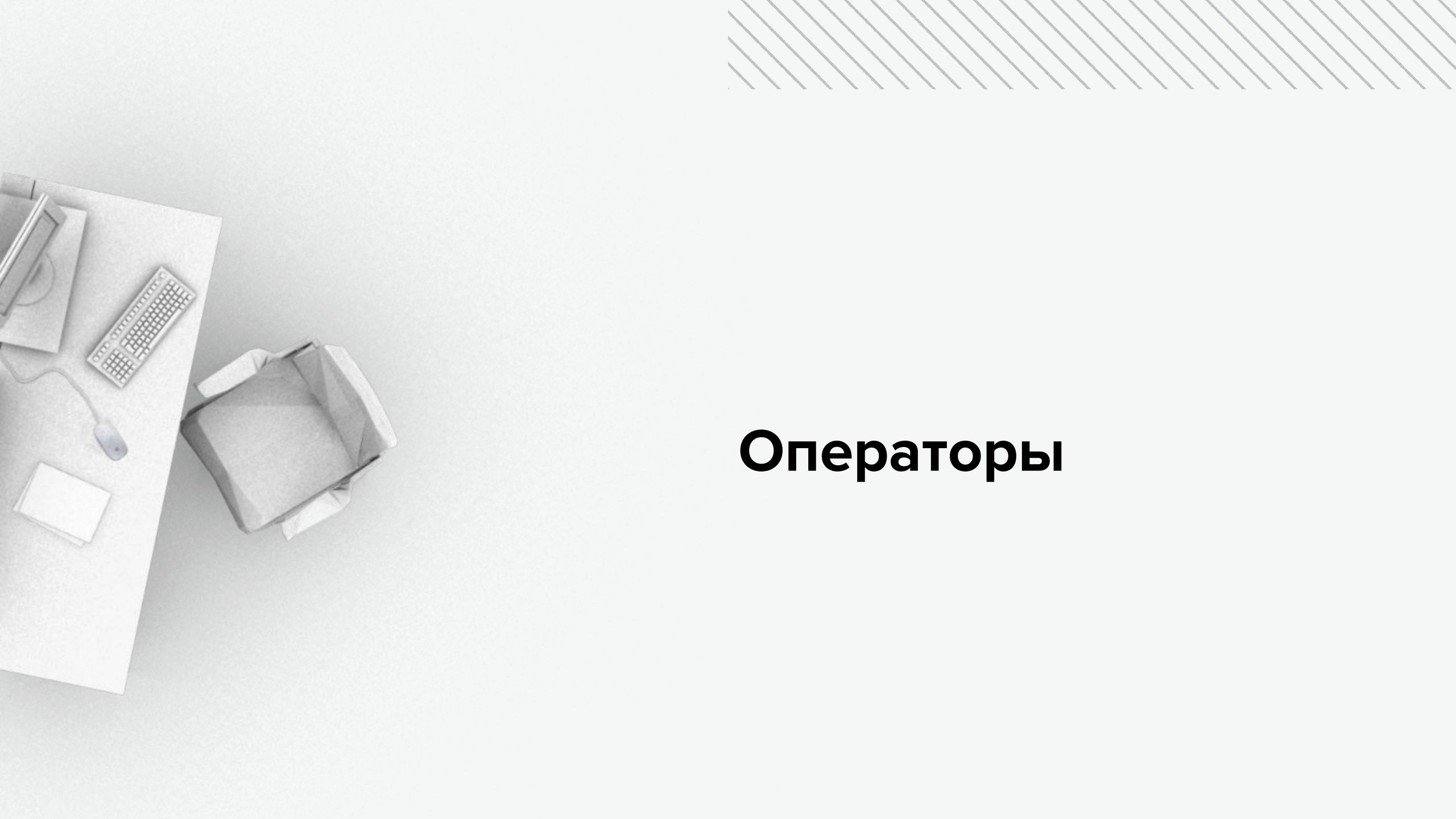
- Когда объект производного класса уничтожается через указатель на базовый класс с неvirtуальным деструктором, то результат не определен;
- Во время исполнения это обычно приводит к тому, что часть объекта, принадлежащая производному классу, никогда не будет уничтожена
- Виртуальный деструктор! При удалении объектов производных классов будет происходить именно то, что нужно.





Абстрактные классы

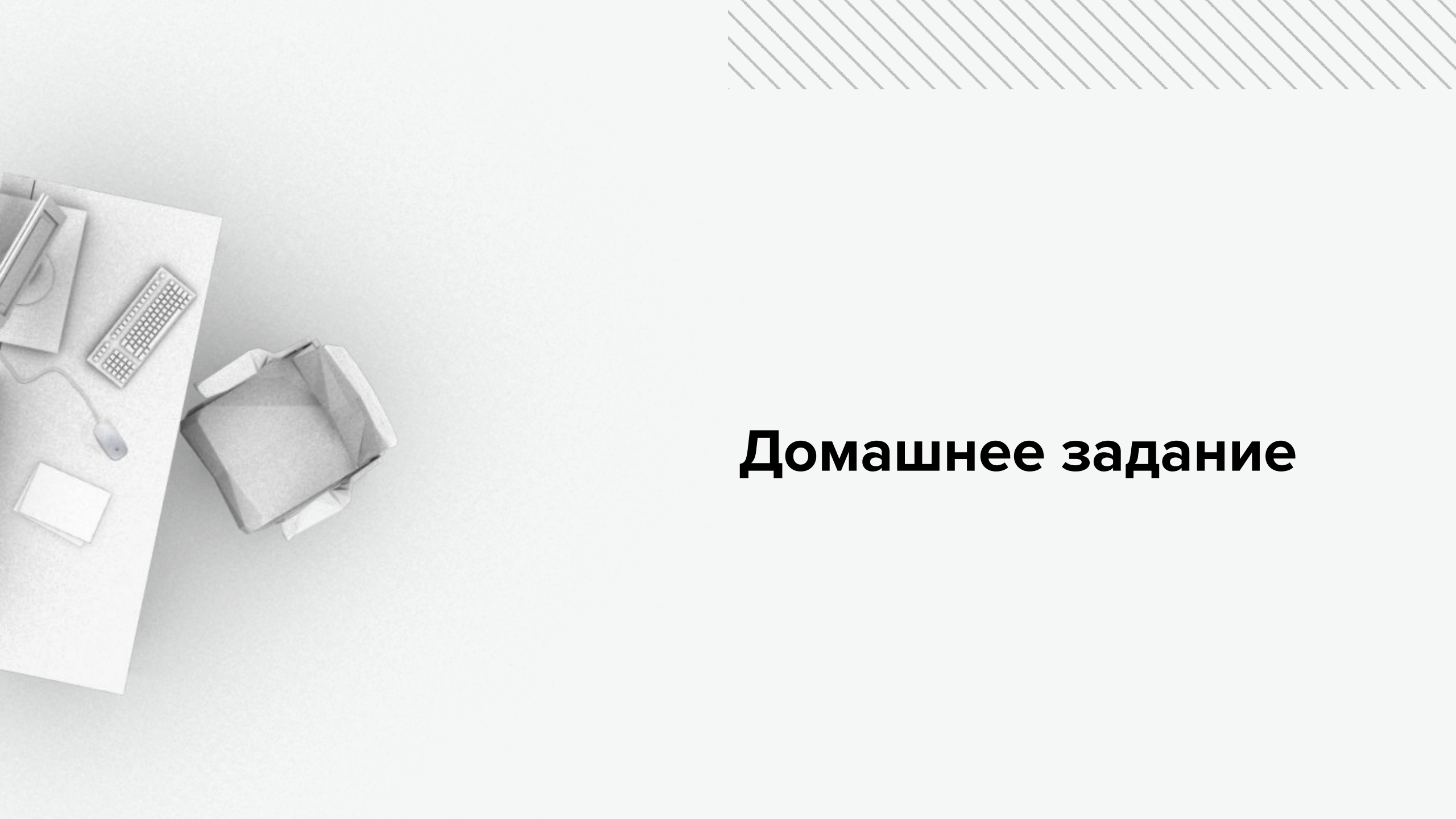
- Класс с одной или несколькими чисто виртуальными функциями называется абстрактным классом;
- Абстрактный класс можно использовать только как интерфейс и в качестве базы для других классов;



Операторы

Операторы

- `bool operator==(const T& other) const`
- `bool operator!=(const T& other) const`
- `T operator+(const T& other) const`
- `T operator-() const`
- `T& operator++() // ++x`
- `T operator++(int) // x++`
- `const T& operator[](size_t i) const`
- Так же есть операторы `new`, `delete` и `,` (запятая)




Домашнее задание

Домашнее задание (1)

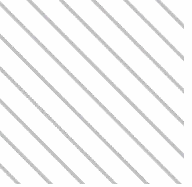
Нужно написать класс-матрицу, тип элементов `int`. В конструкторе задается количество рядов и строк. Поддерживаются операции: получить количество строк(`rows`)/столбцов(`columns`), получить конкретный элемент, умножить на число(`*`), сравнение на равенство/неравенство. В случае ошибки выхода за границы бросать исключение:
`throw std::out_of_range("")`



Домашнее задание (2)



Чтобы реализовать семантику `[]` понадобится прокси-класс. Оператор матрицы возвращает другой класс, в котором тоже используется оператор `[]` и уже этот класс возвращает значение.



Домашнее задание по уроку #4

Домашнее задание №3

#039

?

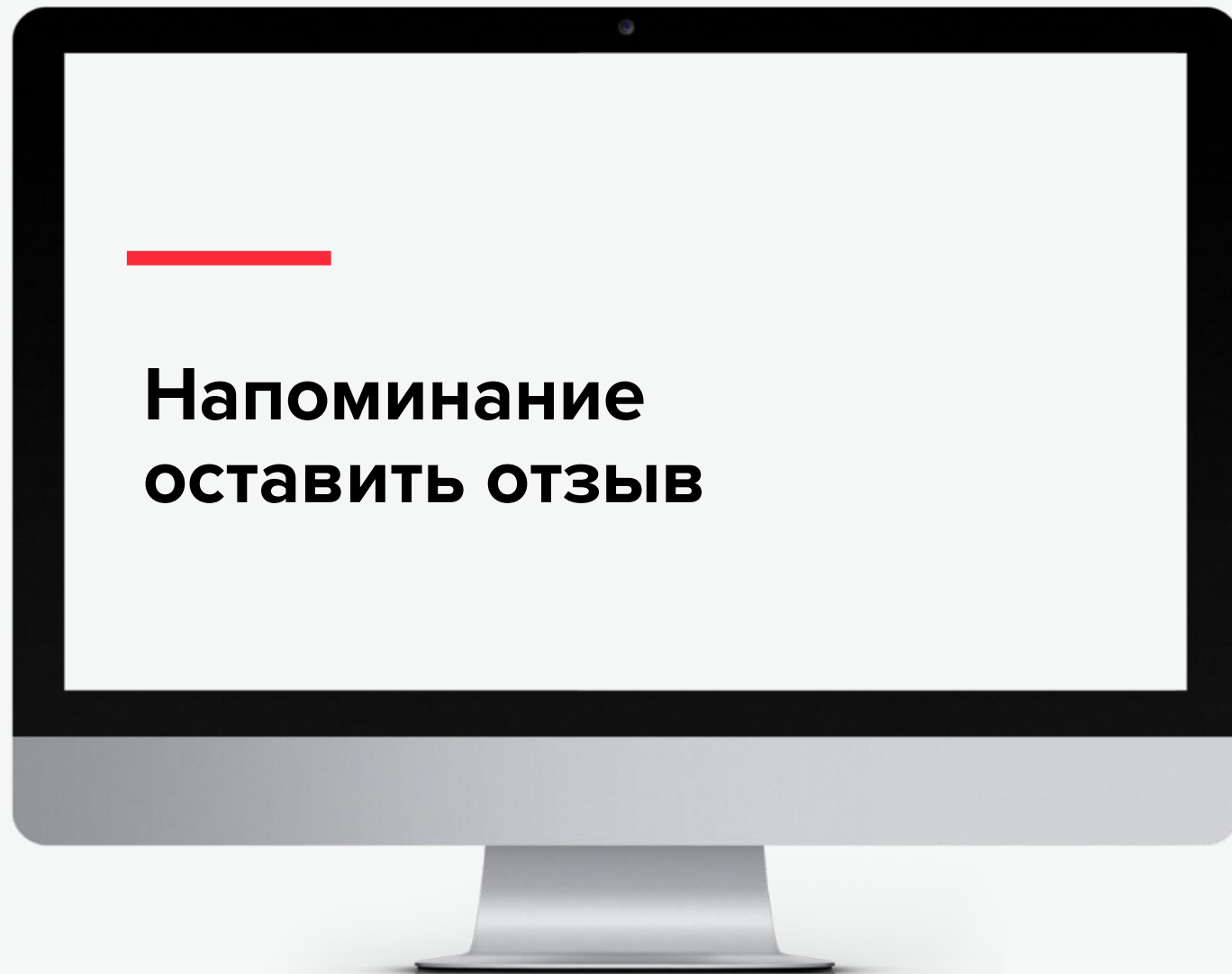
Баллов
за задание

29.10.20

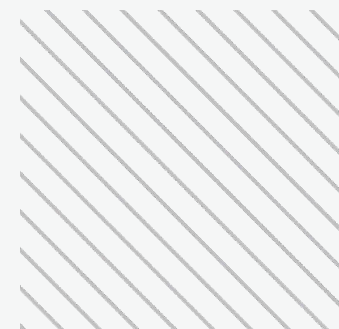
Срок
сдачи

Полезная литература в помощь

- Скотт Мейерс “Эффективный и современный C++”
- Бьерн Страуструп “Языка программирования C++”



**Напоминание
ОСТАВИТЬ ОТЗЫВ**



**СПАСИБО
ЗА ВНИМАНИЕ**

