Объектно-ориентированное программирование

2023

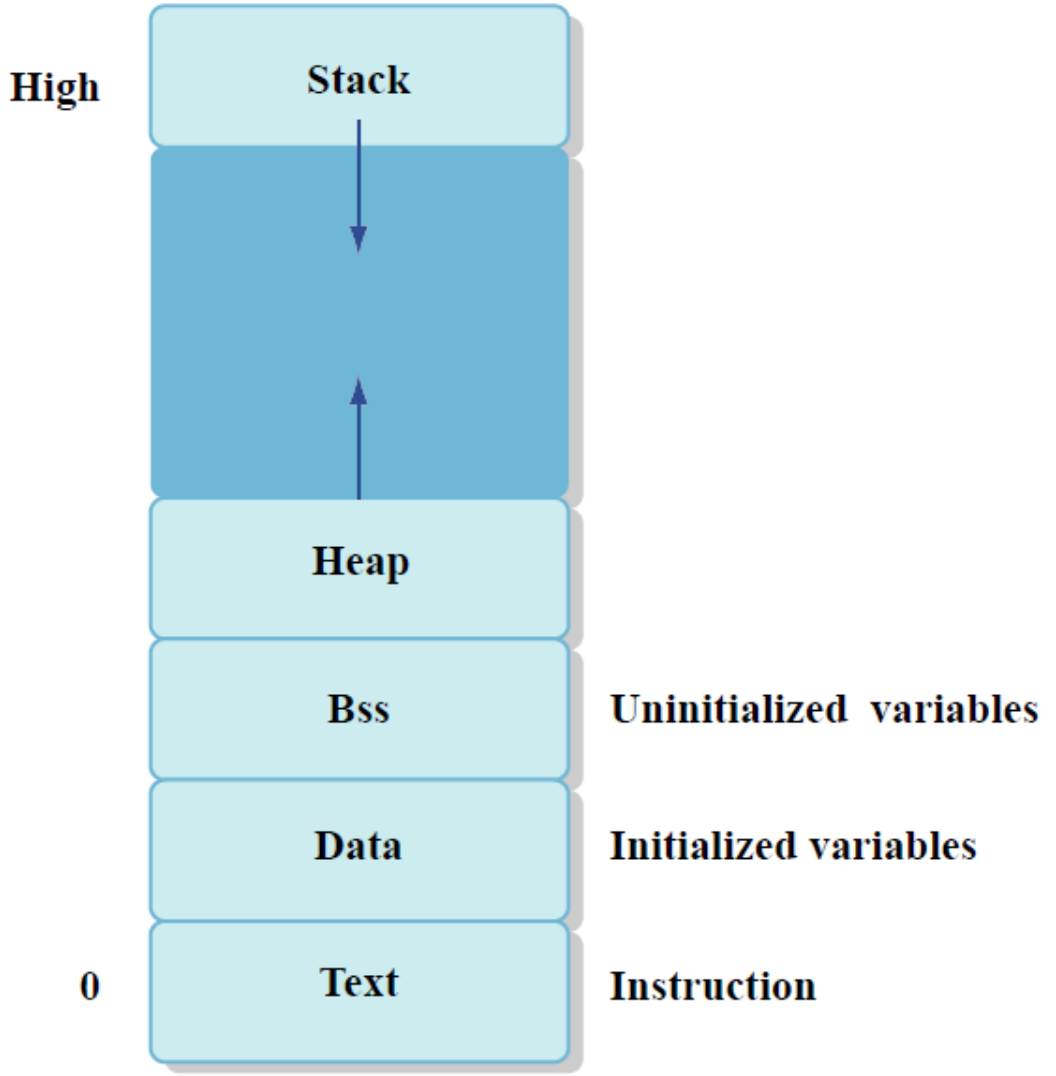
План занятия

- Изучаем работы с памятью
- Указатели
- Массивы
- Ссылки
- Исключения
- Классы

Выделение памяти в стеке (stack)

Функции С размещаются в стеке:

- Функции помещаются в стек, в момент вызова.
- Функции удаляются из стека в момент когда вызывается return.
- Функция может использовать любую память в пределах стека.



Ручное управление памятью в C++

Позволить программе помечать области памяти как «занятые» полезной информацией.

Позволить программы помечать области памяти как «не занятые» по окончании работы. Что бы эти области могли использовать другие алгоритмы и программы.

Управление памятью: Куча (heap)

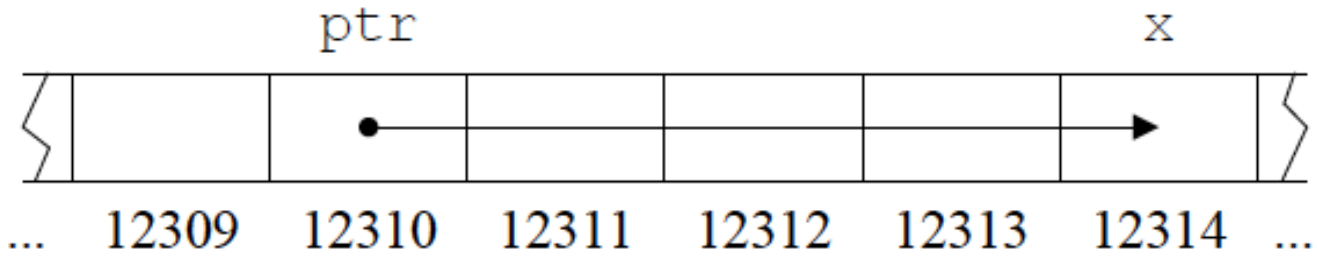
- Куча – это область памяти, который может использовать программа.
- Кучу можно сравнить с гигантским массивом.
- Для работы с кучей используется специальный синтаксис указателей.
- Вся программа может получить доступ к куче.

Addr.	Contents
⋮	⋮
0xbee	0xbeef
0xbf4	0xfeed
⋮	⋮

Указатель – это число

Указатель хранит адрес переменной!

```
int x = 5;
int * ptr = &x;
```



Пустой указатель **nullptr**

- Раньше, для обнуления указателей использовался макрос NULL, являющийся нулем — целым типом, что, естественно, вызывало проблемы (например, при перегрузке функций).
- Ключевое слово **nullptr** имеет свой собственный тип **std::nullptr_t**, что избавляет нас от бывших проблем.
- Существуют неявные преобразования nullptr к нулевому указателю любого типа и к bool.

Примеры

01_DeclaringAndUsingPointers

02_PointerToChar

03_ConstPointerAndPointerToConst

04_PointersAndArrays

05_PointerArithmetic_Navigation

06_PointerArithmetic_DistanceBetweenElements

07_DynamicMemoryAllocation

08_DanglingPointers

09_WhenNewFails

10_MemoryLeaks

11_DynamicallyAllocatedArrays

Передача информации о переменных в функции и объекты

Из С мы помним, что передавать переменные можно:

- - «по значению» – путем копирования;
- - «с помощью указателя» – тогда копируется указатель, а переменная «остается на месте»;

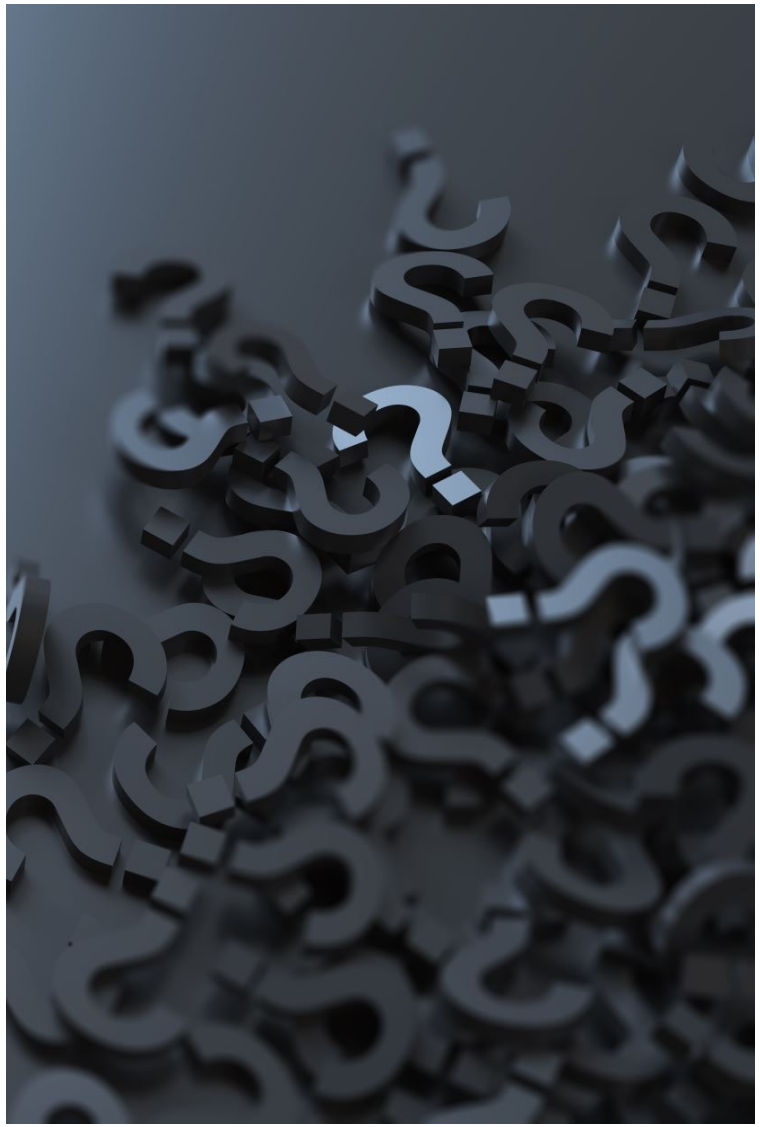
&ссылки

// структура объявления ссылок

/*тип*/ &/*имя ссылки*/ = /*имя переменной*/;

& Ссылки

- Ссылка — это **синоним** имени переменной, т. е. другое имя для использования переменной.
- Отличие ссылки от указательной переменной в том, что ссылка **не является объектом**. Для названия ссылки может не отводиться место в памяти. Для указательной переменной место в памяти выделяется всегда.
- Ссылка **не может ссылаться на несуществующий объект**, указатель может.
- Ссылку нельзя переназначить, а указательную переменную можно.



Lvalue & Rvalue переменные

С каждой **обычной** переменной связаны две вещи – **адрес** и **значение**.

int I; // создать переменную по адресу, например 0x10000

I = 17; // изменить значение по адресу 0x10000 на 17

А что будет если у меня есть только значение? Могу ли я сделать так: 20=10; ?

с любым выражением связаны либо адрес и значение, либо только значение

- Для того, чтобы отличать выражения, обозначающие объекты, от выражений, обозначающих только значения, ввели понятия **lvalue** и **rvalue**.
- Изначально слово lvalue использовалось для обозначения выражений, которые могли стоять слева от знака присваивания (*left-value*); им противопоставлялись выражения, которые могли находиться только справа от знака присваивания (*right-value*).

i	—	lvalue
++i	—	lvalue
*&i	—	lvalue
a[5]	—	lvalue
a[i]	—	lvalue
10	—	rvalue
i + 1	—	rvalue
i++	—	rvalue

Пример

Примеры

12_DeclaringAndUsingReferences

13_ComparingPointersAndReferences

14_ReferencesAndConst

15_ReferencesWithRangeBasedForLoops

16_LvalueAndRValue

Как обрабатывать алгоритмические ошибки?

1. Вернуть результат операции явно (операция успешна / операция не успешна)
2. Вернуть результат выполнения операции как один из параметров (по ссылке)

Пример

17_ReturnError

Exceptions

Для реализации механизма обработки исключений в язык Си++ введены следующие три ключевых (служебных) слова:

1. **try** (контролировать)
2. **catch** (ловить)
3. **throw** (генерировать, порождать, бросать, посылать, формировать).

Exception

Исключение это:

1. Объект, наследник класса std::exception (#include <exception>)
2. Событие, прерывающее обработку программы.

Под прерыванием мы понимаем, что срабатывание исключение, аналогично срабатыванию оператора **return**.

Но есть существенное отличие: **return** возвращает результат в то место, где вызвали функцию.

Исключение возвращает объект исключения только в те места, где его явно ловят (**catch**)!

И только если исключение сработало (**throw**) в месте где мы его контролируем (**try**)!

Если исключение не поймать (**catch**) то оно будет прерывать работу функций, поднимаясь вверх по стеку вызова, пока не остановит программу.

Try / Catch

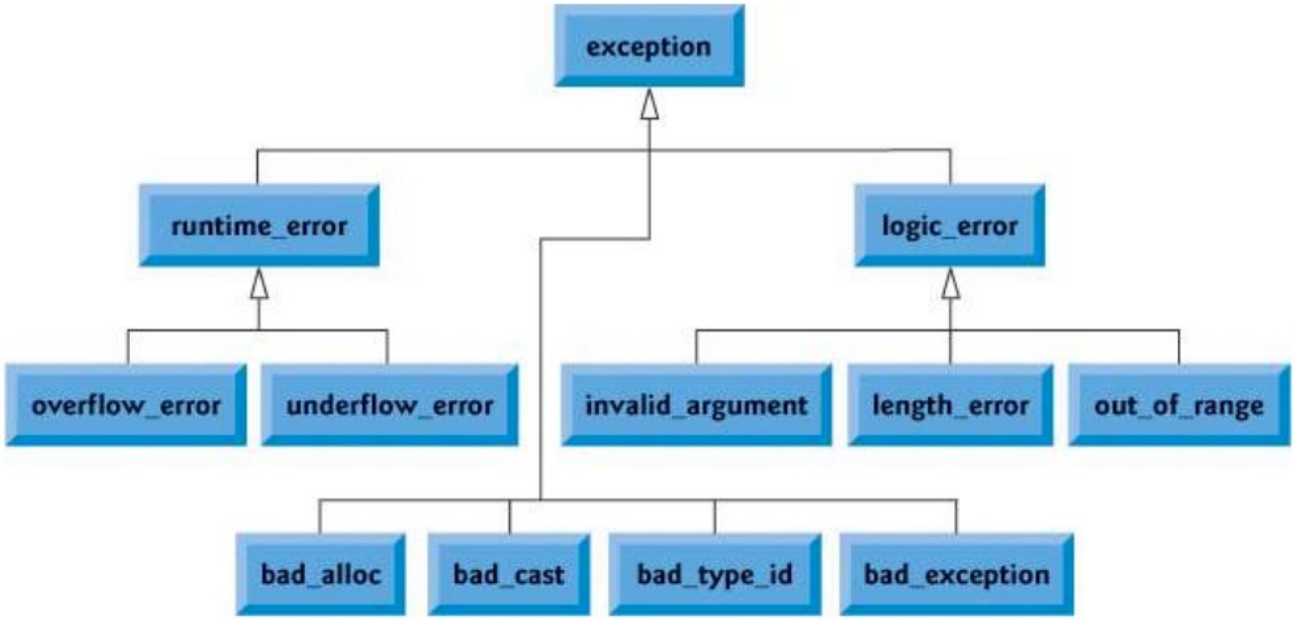
- Служебное слово try позволяет выделить в любом месте исполняемого текста программы так называемый контролируемый блок:

```
try {
//операторы
//операторы
} catch (Тип_исключения1 имя){
//операторы
} catch (Тип_исключения2 имя){
//операторы
}
```

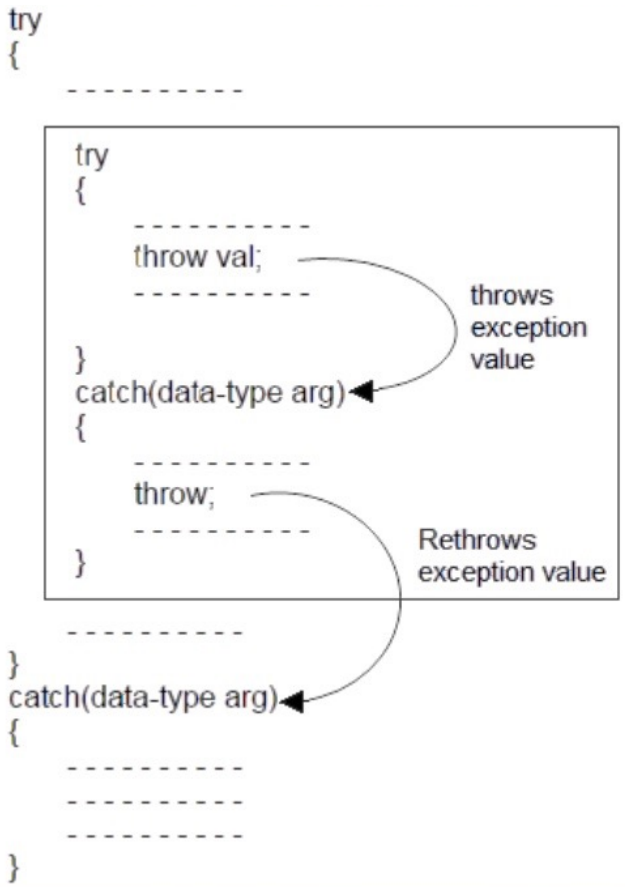
аргумент исключения – это все что угодно

- C++ позволяет создавать исключения любого типа, хотя обычно рекомендуется создавать типы, производные от **std::exception**. Исключение в C++ может быть перехвачено обработчиком **catch**, в котором определен тот же тип, что и у созданного исключения, или обработчиком, который способен перехватывать любой тип исключения.
- Если созданное исключение имеет тип класса, у которого имеется один или несколько базовых классов, то его могут перехватывать обработчики, которые принимают базовые классы (и ссылки на базовые классы) этого типа исключения.
- Обратите внимание, что если исключение перехватывается по ссылке, то оно привязывается к самому объекту исключения; в противном случае обрабатывается его копия (как и в случае с аргументами функции).

Стандартные исключения C++ #include <exception>



Повторное «возбуждение» исключений



Exception Ptr

- **std::current_exception** — данная функция возвращает exception_ptr. Если мы находимся внутри блока catch, то возвращает exception_ptr, который содержит обрабатываемое в данный момент текущим потоком исключение, если вызывать ее вне блока catch, то она вернет пустой объект exception_ptr
- **std:: rethrow_exception** — данная функция бросает исключение, которое содержится в exception_ptr. Если входной параметр не содержит исключения (пустой объект), то результат не определен.
- **std:: make_exception_ptr** — данная функция, может сконструировать exception_ptr без бросания исключения.

Пример

18_ComplexException

Внимание!

- Помни, что в блоке catch могут возникать свои исключения!
- Если в блоке catch идет высвобождение ресурсов (удаление объектов, закрытие дескрипторов файлов) то их надо самих помещать в еще один вложенный блок try/catch.

Пример

19_ExceptionInCatch

Обработка всех исключений

```
try {
    throw CSomeOtherException();
}
catch(...) {
    // Catch all exceptions – dangerous!!!
    // Respond (perhaps only partially) to the exception, then
    // re-throw to pass the exception to some other handler
    // ...
    throw;
}
```

Можно ли использовать exception как return «на стероидах»?

20_ExFast

noexcept

- В стандарте ISO C++11 был представлен оператор **noexcept**. Он поддерживается в Visual Studio 2015 и более поздних версий. Когда это возможно, используйте noexcept, чтобы задать возможность вызова функцией исключений.
- В предыдущих версиях стандарта можно использовать **throw();**
- **Если метод с noexcept все таки сгенерирует исключение, то оно перехвачено уже не будет.**

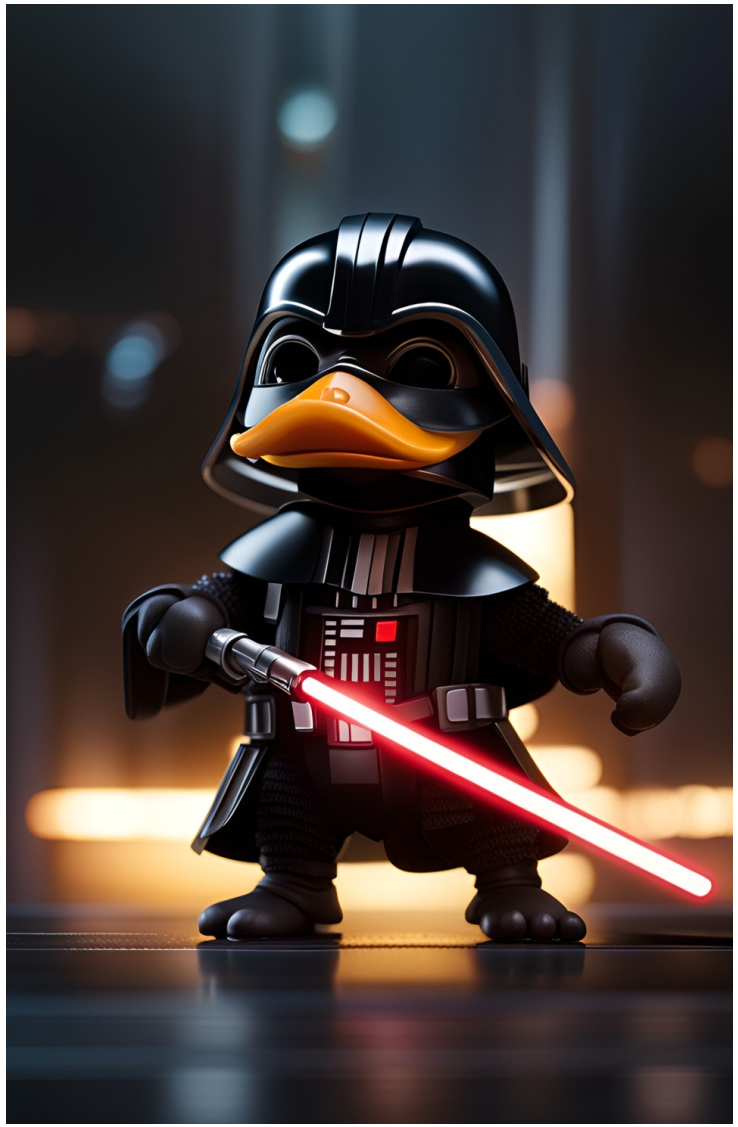
Пример

21_Noexcept

Exceptions
итого

1. Помогает создать надежную программу;
2. Отделяет код обработки ошибок от основной логики программы;
3. Обработка исключений может быть реализована за пределами основного кода программы;
4. Существует возможность обрабатывать только выбранные типы исключений;
5. Программа, обрабатывающая исключения не остановится без объяснения причин (например, вывода на экран причины возникновения исключений);

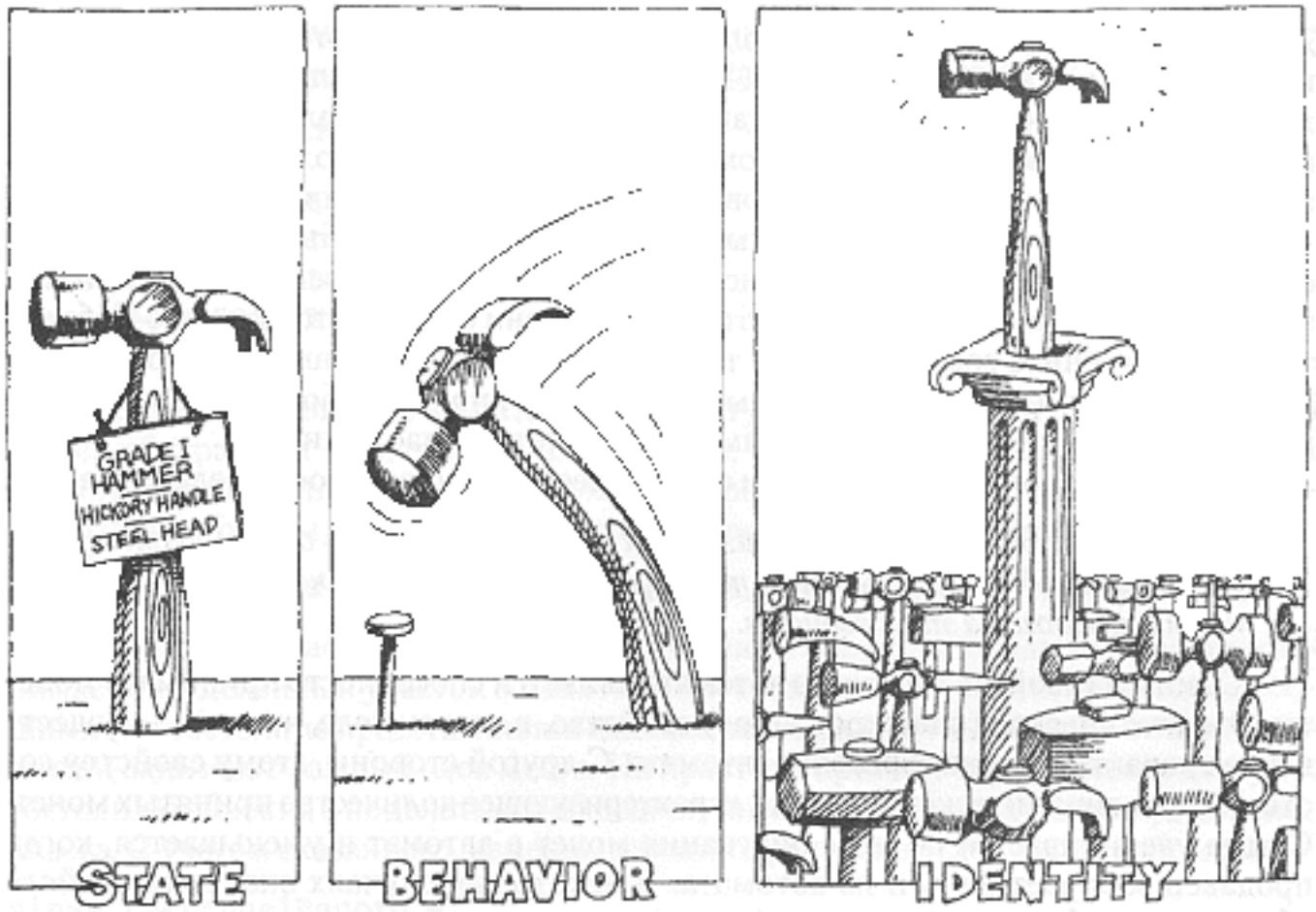
Классы и объекты



Объект

- «Объект представляет собой конкретный опознаваемый предмет, единицу или сущность (реальную или абстрактную), имеющую четко определенное функциональное назначение в данной предметной области»

- Smith, M. and Tockey, S. 1988. An Integrated Approach to Software Requirements Definition Using Objects. Seattle, WA: Boeing Commercial Airplane Support Division, p.132.



Свойства объекта

Состояние
в любой момент времени объект находится в каком-либо состоянии, которое можно измерить / сравнить / скопировать

Поведение
объект может реагировать на внешние события либо меняя свое состояние, либо создавая новые события

Идентификация
объект всегда можно отличить от другого объекта

Класс

1. **Определение.**
   Классом будем называть группу объектов, с общей структурой и поведением.
2. Смысл программы на C++ это описание классов!
3. Даже если нужен всего один объект – мы будем описывать класс.

Очень простой класс объектов

```
class MyClass
{
public:
    int Number;
    void doSomething();
};
```

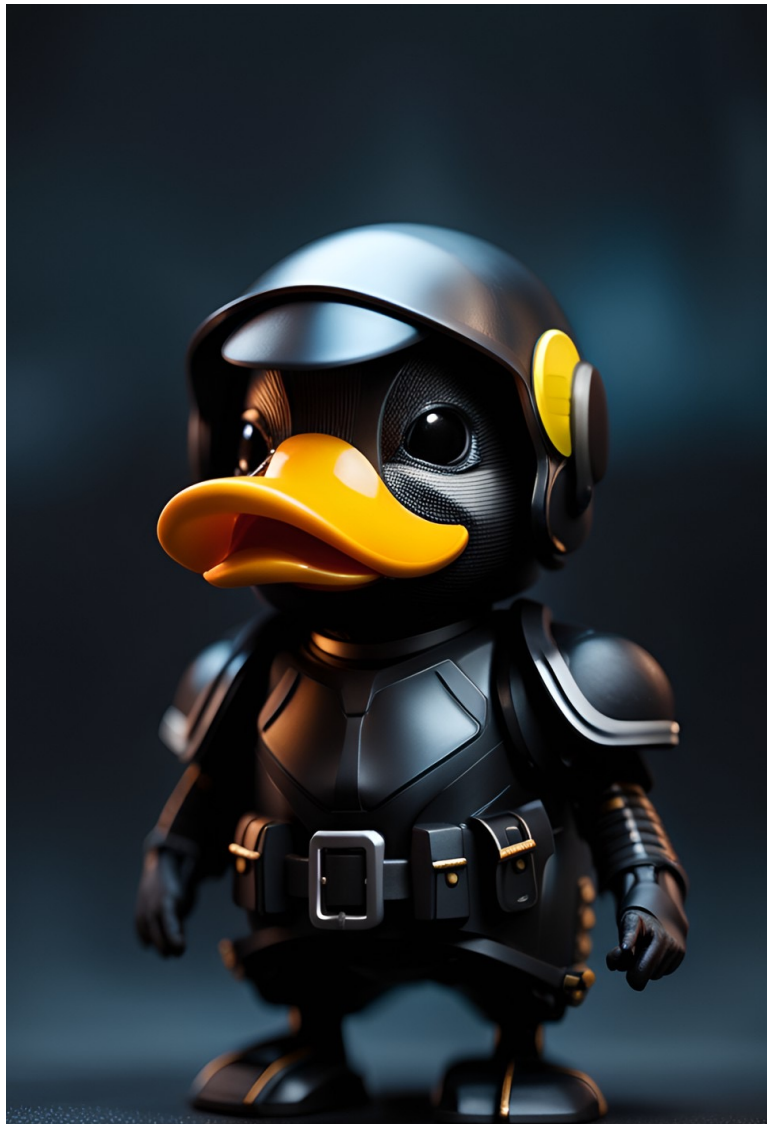
- **class** – ключевое слово
- **public** – область видимости атрибутов и методов класса

`int Number` – атрибут класса

`void doSomething()` - метод класса

Примеры

22_YourFirstClass
23_Constructors
24_DefaultedConstructors
25_SettersAndGetters
26_ClassAcrossMultipleFiles
27_ManagingClassObjectsThroughPointers
28_Destructors
29_OrderOfConstructorDestructorCalls
30_ThisPointer
31_Struct
32_SizeOfClassObjects
33_ConstMember



Инкапсуляция, пример: контроль доступа в C++

Член класса может быть **частным** (**private**), **защищенным** (**protected**) или **общим** (**public**):

1. Частный член класса X могут использовать только функции-члены и друзья класса X.
2. Защищенный член класса X могут использовать только функции-члены и друзья класса X, а так же функции-члены и друзья всех производных от X классов (рассмотрим далее).
3. Общий член класса можно использовать в любой функции.

Контроль доступа применяется единообразно ко всем именам. На контроль доступа не влияет, какую именно сущность обозначает имя.

Друзья класса объявляются с помощью ключевого слова **friend**. Объявление указывается в описании того класса, к частным свойствам и методам которого нужно подучать доступ.

Конструктор

Если у класса есть конструктор, он вызывается всякий раз при создании объекта этого класса. Если у класса есть деструктор, он вызывается всякий раз, когда уничтожается объект этого класса.

Объект может создаваться как:

1. автоматический, который создается каждый раз, когда его описание встречается при выполнении программы, и уничтожается по выходе из блока, в котором он описан;
2. статический, который создается один раз при запуске программы и уничтожается при ее завершении;
3. объект в свободной памяти, который создается операцией new и уничтожается операцией delete;
4. объект-член, который создается в процессе создания другого класса или при создании массива, элементом которого он является.

Сколько раз вызовется конструктор?

```
struct Integer {
    int val;
    Integer() {
        val = 0;
        std::cout << "default constructor" <<
        std::endl;
    }
};


int main() {
 Integer arr[3];
}
```

Константные функции

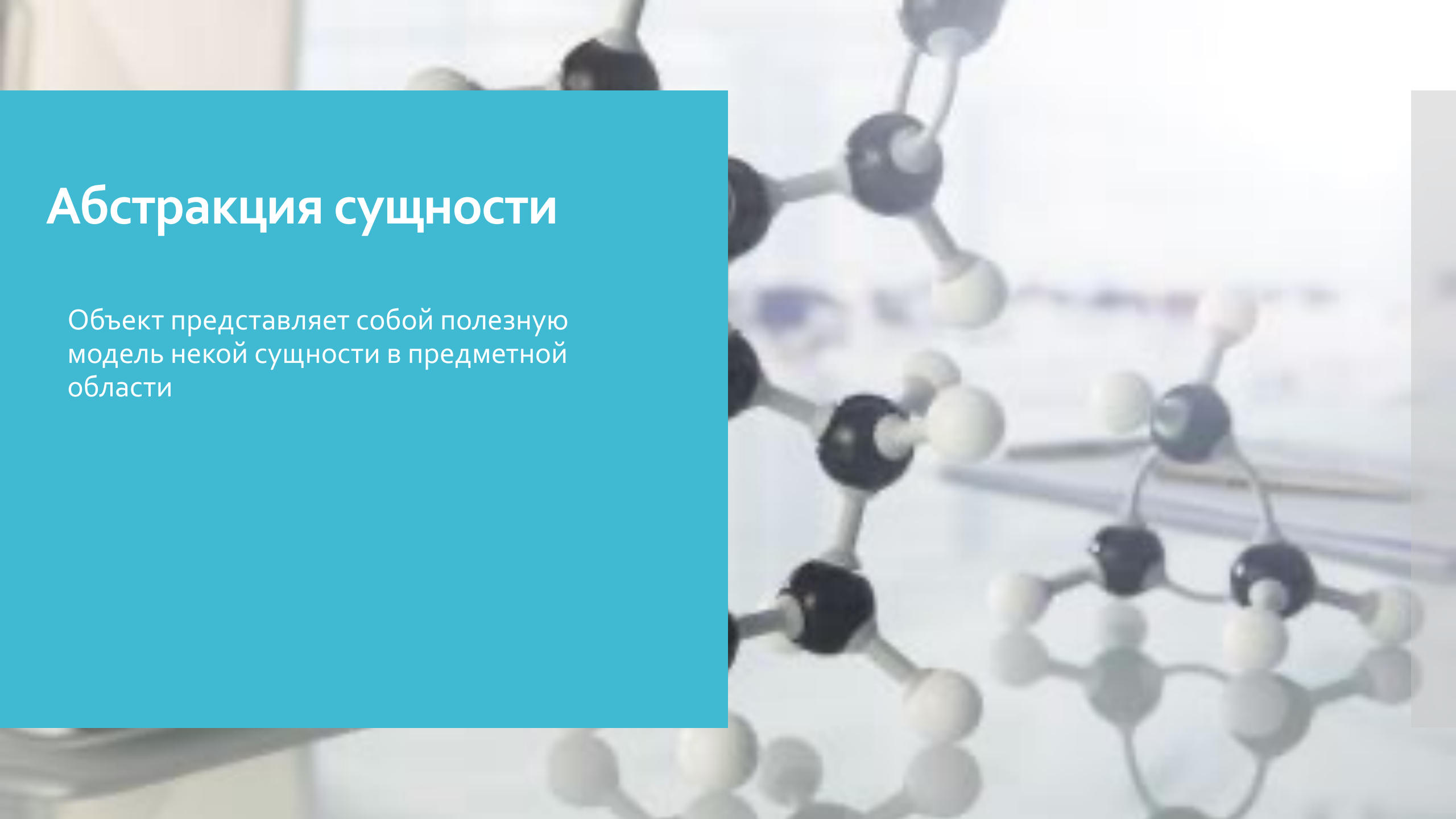
```
struct A {
        int x;
        void f(int a) const {
                x = a; // <-- не работает
        }
};
```

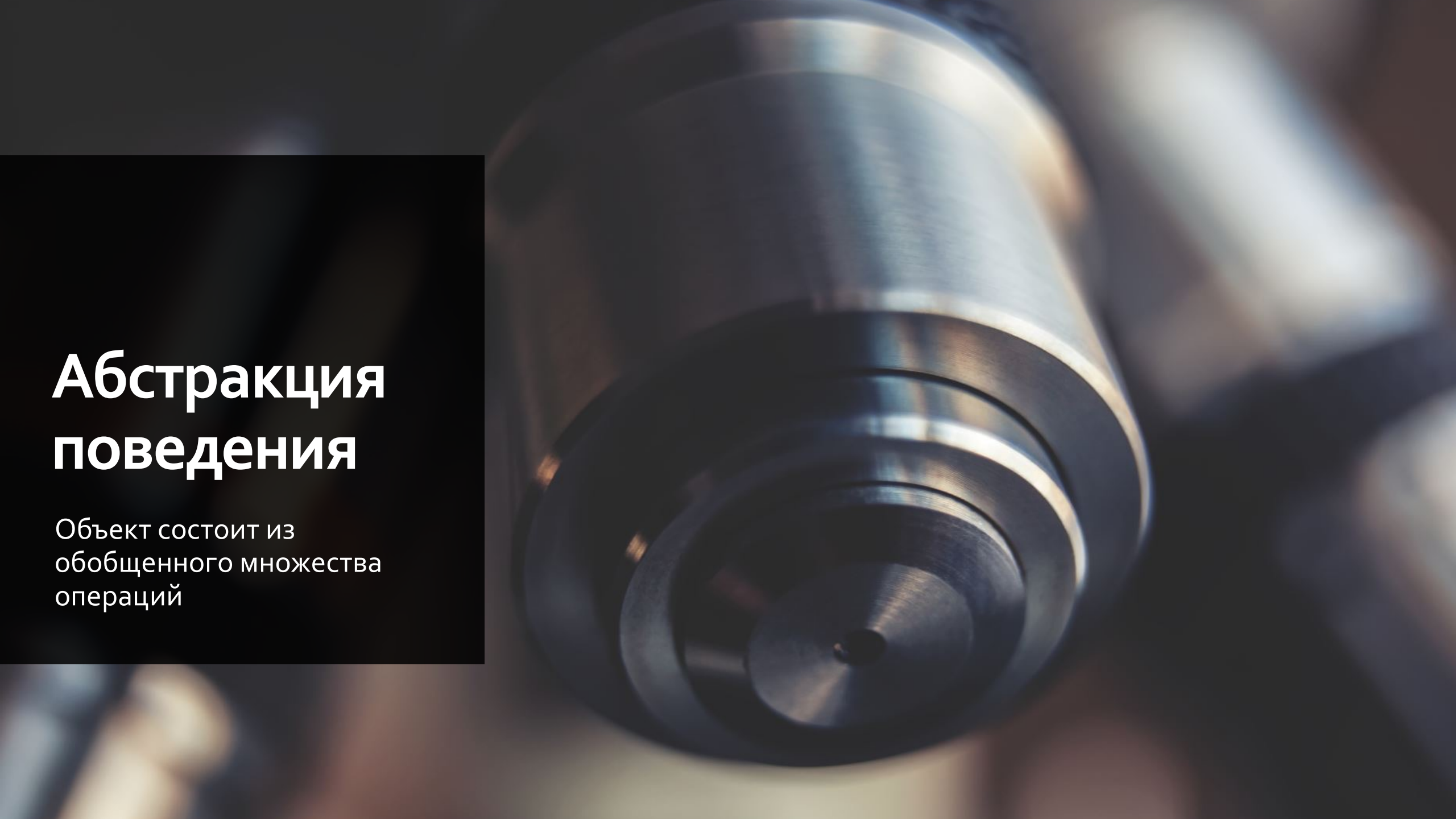
Как придумать класс?

- *Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя.*

Абстракция сущности

Объект представляет собой полезную модель некой сущности в предметной области





Абстракция поведения

Объект состоит из обобщенного множества операций

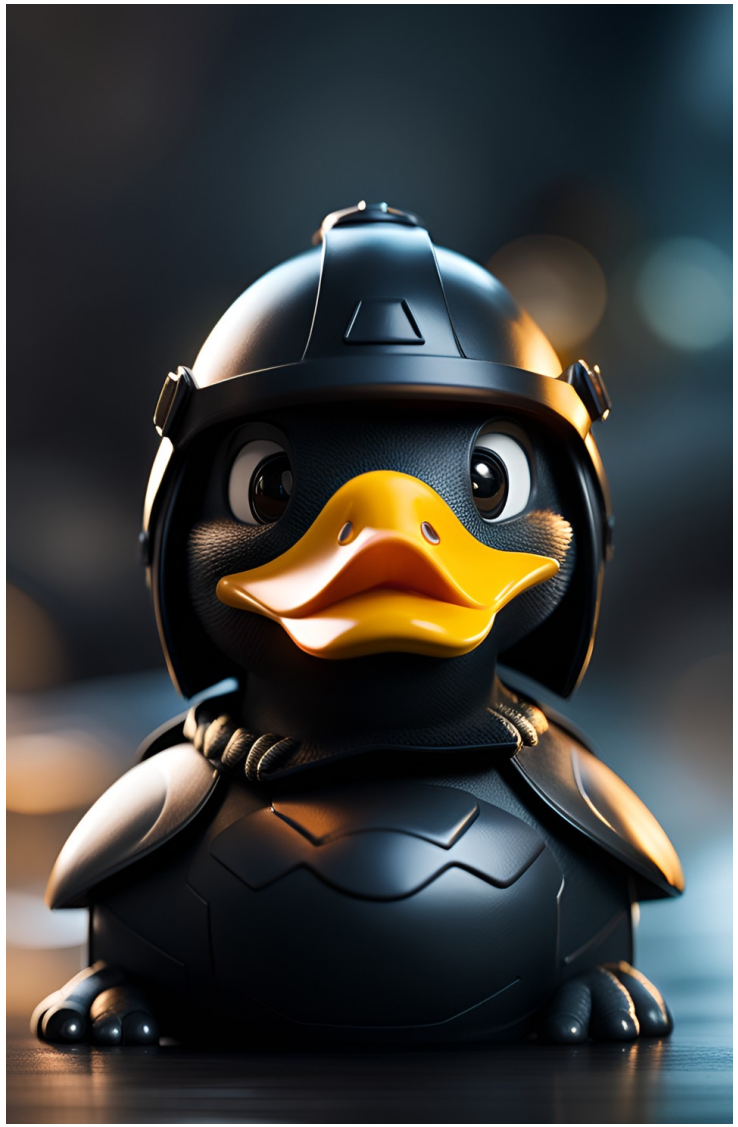
Абстракция виртуальной машины

Объект группирует операции, которые либо вместе используются более высоким уровнем управления, либо сами используют некоторый набор операций более низкого уровня

Произвольная абстракция

Объект включает в себя набор операций, не имеющих друг с другом ничего общего

Лабораторная работа №2



Спасибо!

На сегодня все