ООП:
абстракция и инкапсуляция

Максим Бакиров
C++ - Разработчик в Яндекс

Проверка связи

Поставьте “+”, если меня видно и слышно

Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включен звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти

Максим Бакиров

О спикере:

- В C++ разработке с 2017 года
- С 2019 года работает в команде разработки Яндекс Браузера

Вспоминаем прошлое занятие

Вопрос: что такое класс?

Вспоминаем прошлое занятие

Вопрос: что такое класс?

Ответ: класс - это сущность, объединяющая данные и операции над этими данными

Вспоминаем прошлое занятие

Вопрос: что такое объект?

Вспоминаем прошлое занятие

Вопрос: что такое объект?

Ответ: объект - это экземпляр класса

Вспоминаем прошлое занятие

Вопрос: какие члены класса вы знаете?

Вспоминаем прошлое занятие

Вопрос: какие члены класса вы знаете?

Ответ: поля, методы, конструкторы

Вспоминаем прошлое занятие

Вопрос: что такое метод и как его вызвать?

Вспоминаем прошлое занятие

Вопрос: что такое метод и как его вызвать?

Ответ: метод - это функция, объявленная внутри класса. Чтобы вызвать метод, нужно после переменной, содержащей объект класса, поставить точку, написать имя метода и передать туда параметры

Вспоминаем прошлое занятие

Вопрос: что такое конструктор и как его вызвать?

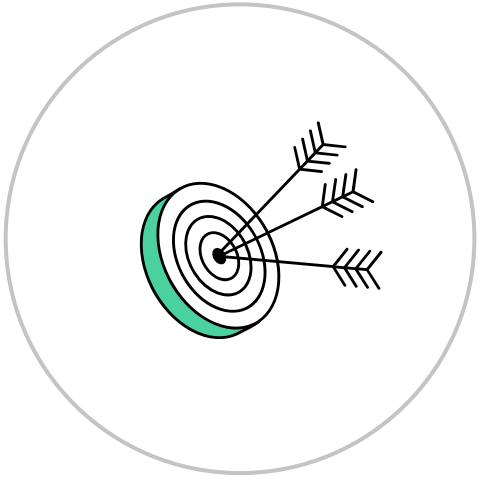
Вспоминаем прошлое занятие

Вопрос: что такое конструктор и как его вызвать?

Ответ: конструктор - это специальный код внутри класса, который вызывается при создании объекта

Цели занятия

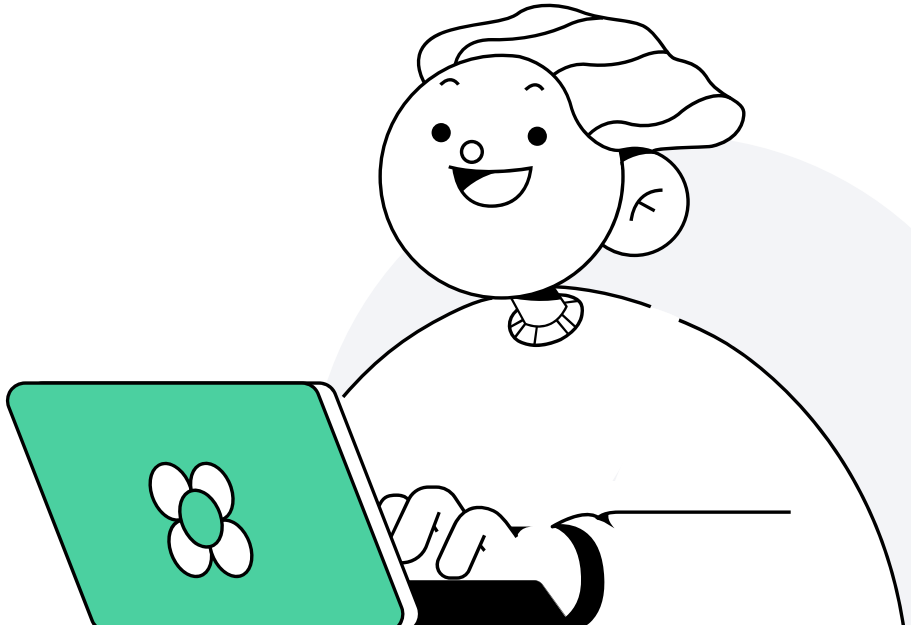
- Разберёмся, что такое ООП
- Познакомимся с историей ООП
- Узнаем, что такое абстракция
- Выясним, что такое инкапсуляция



План занятия

1. ООП
2. Абстракция
3. Инкапсуляция
4. Итоги
5. Домашнее задание

*Нажми на нужный раздел для перехода



ООП

1

Что такое ООП?

Вопрос: Мы с вами уже встречались с термином ООП, когда говорили про классы. Кто вспомнит, как это расшифровывается?

ООП

Объектно-ориентированное программирование

это парадигма программирования

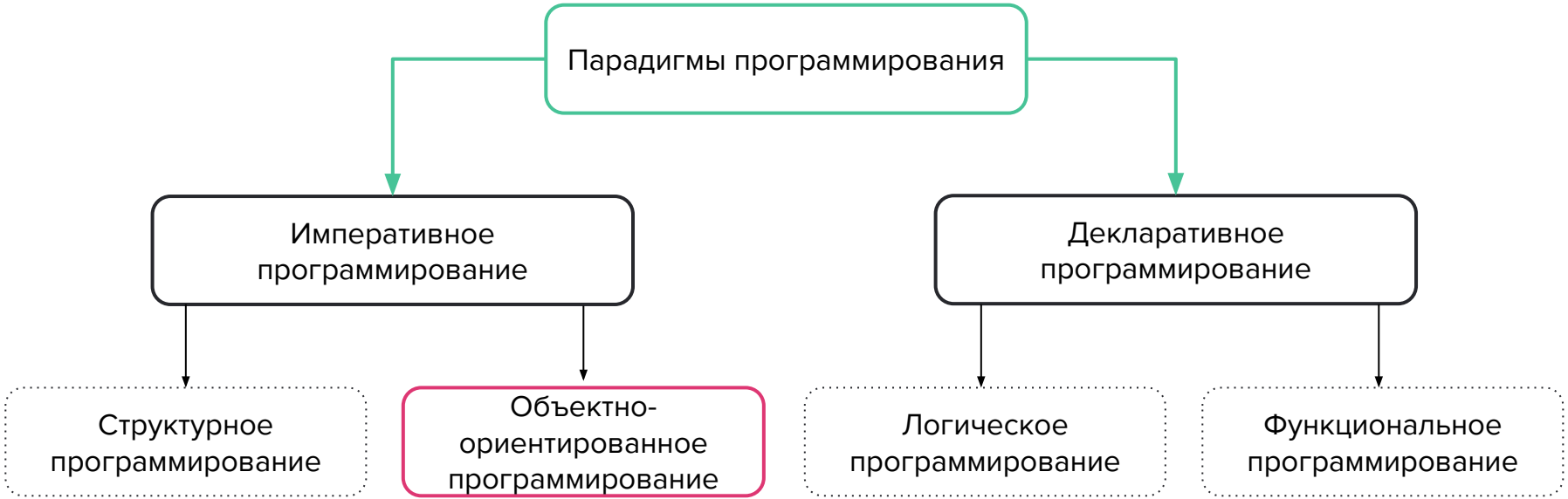
Парадигма программирования - это метод организации программ

Парадигма программирования определяет:

- Какие “строительные блоки” - компоненты - мы используем при написании программ
- Как мы структурируем наши программы
- Как мы связываем наши компоненты, как объединяем в единую систему

Какие бывают парадигмы программирования

ООП - не единственная парадигма программирования

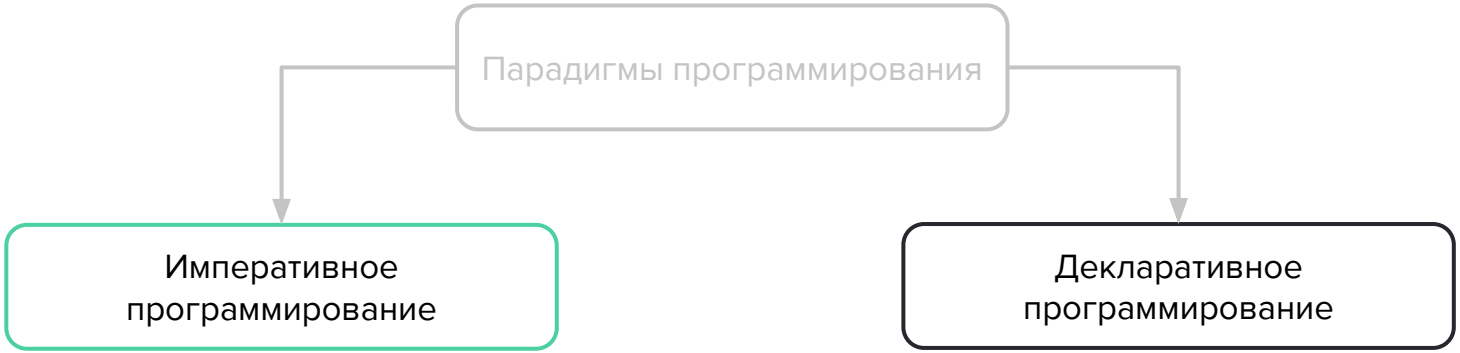


Императивная и декларативная парадигмы

Код в императивной парадигме описывает, **как** решить поставленную перед нами задачу

Код в декларативной парадигме программирования, описывает **задачу**, которую нам надо решить. А среда, интерпретирующая код, уже решает, как это сделать.

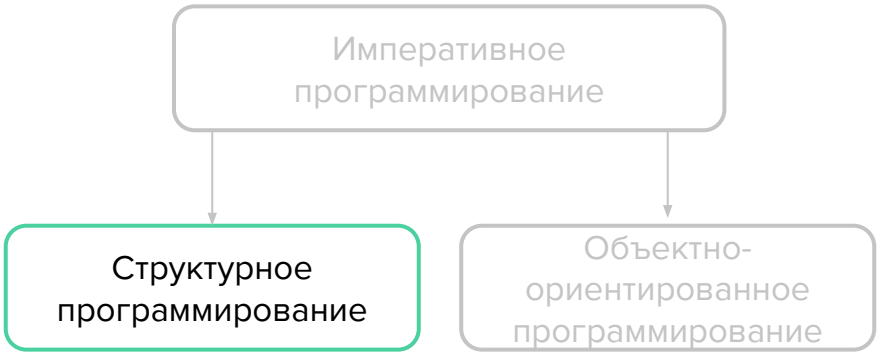
Мы будем разбирать императивные парадигмы



Структурная парадигма

До сих пор мы писали в **структурной** парадигме программирования (иногда её ещё называют **процедурной**). В ней основными “строительными блоками” являются **функции**, которые, в свою очередь, состоят из команд

Функции выполняют разные задачи, вызывают друг друга, и таким образом мы строим программы



Объектно-ориентированная парадигма (ООП)

В рамках объектно-ориентированной парадигмы основным “строительным блоком” является не функция, а **объект**

Объект, в свою очередь - это **экземпляр класса**

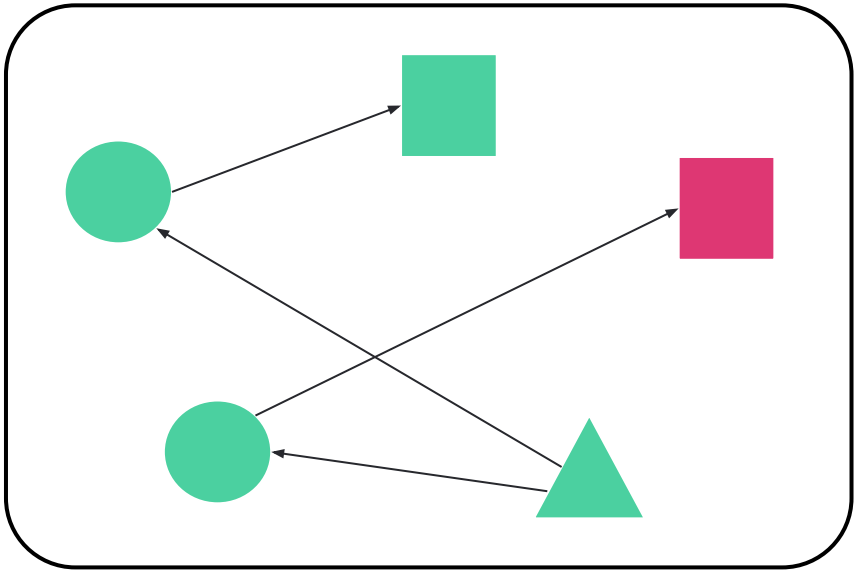
А класс - это шаблон, который описывает **структуру** (из чего состоит) и **поведение** (что умеет) объект

Объектно-ориентированная парадигма (ООП)

Программа, написанная в объектно-ориентированной парадигме, представляет собой множество объектов, общающихся друг с другом

Объекты, могут быть экземплярами **разных** классов

Одни объекты создают другие, обращаются к полям и методам других объектов



Парадигма и язык

Важно понимать, что конкретная парадигма программирования может не поддерживаться конкретным языком программирования

В то же время, один язык программирования может поддерживать сразу несколько парадигм программирования

Парадигма и язык

А это значит, что просто создавая программы на определенном языке, вы не обязательно пишете код в какой-то конкретной парадигме - это зависит от вас

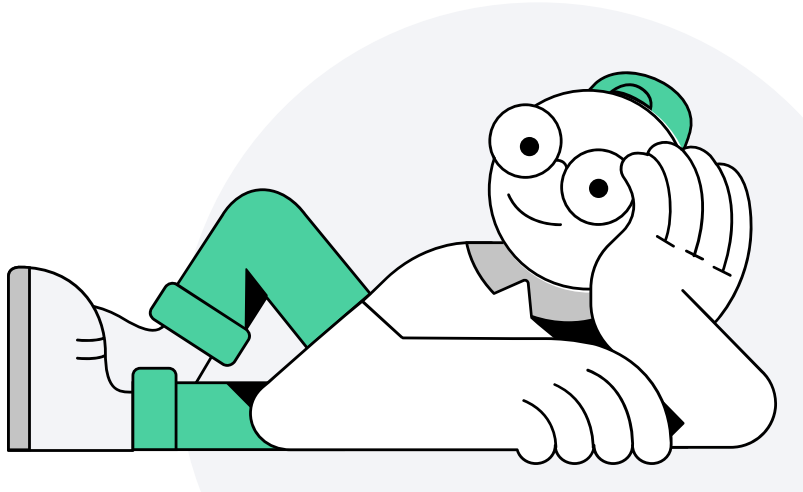
Например, язык C++ поддерживает несколько парадигм:

- структурную
- объектно-ориентированную
- отчасти функциональную

Зачем нужно ООП

Зачем вообще придумали классы? Кому это понадобилось?

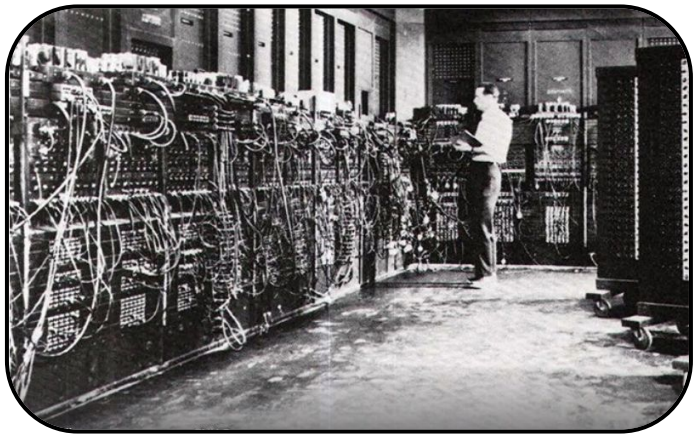
Чтобы ответить на этот вопрос, нам нужно немного погрузиться в историю

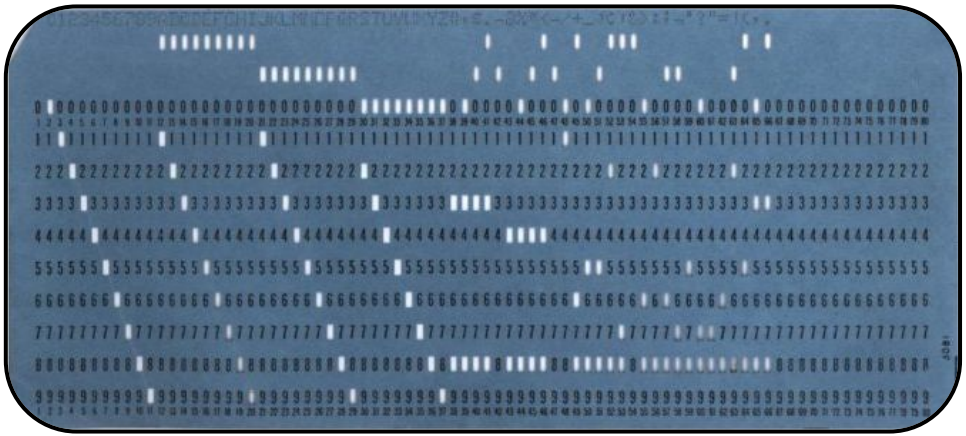


Сначала было...

Когда люди только начали писать программы, их возможности были очень сильно ограничены - компьютеры были огромными и занимали целые комнаты, скорость их работы по сравнению с сегодняшней была экстремально низкой

А программы вносились в компьютеры с помощью физических носителей - перфокарт





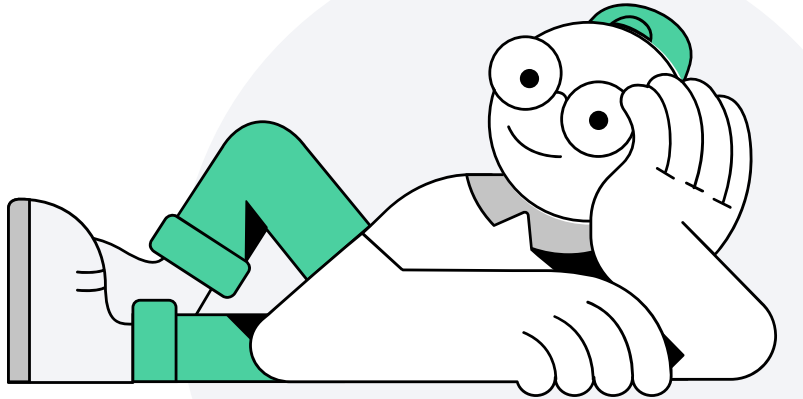
Сначала было...

В таких условиях много кода не напишешь - люди составляли совсем простые по современным мерками программы, и для этого было достаточно одного человека

Именно так появилась парадигма программирования, которая впоследствии была названа **императивной** - программа состоит просто из набора команд, которые последовательно выполняются друг за другом

Немного позже

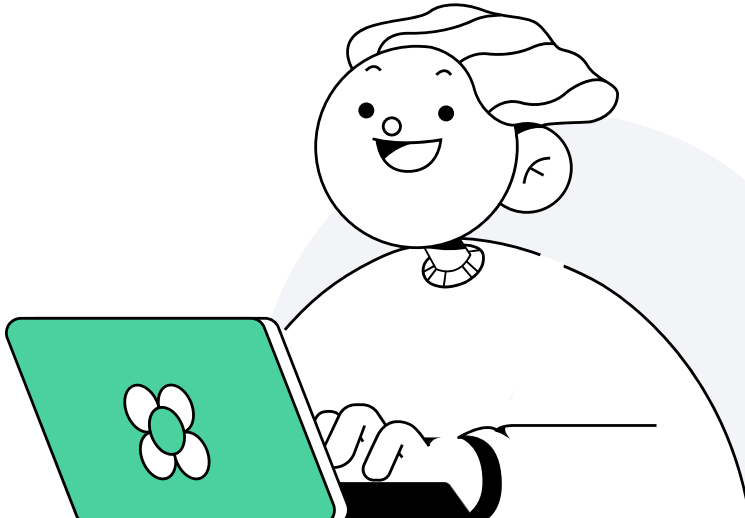
После того, как стало возможным писать программы на самом компьютере, когда скорость вычислений выросла - стало возможным создавать более сложные программы. Более сложные программы - это больше выполняемых задач и, соответственно, больше кода



Немного позже

Когда кода стало много (сотни и тысячи строк), люди поняли, что его надо как-то разбивать на более простые компоненты и структурировать. Так появились функции (процедуры), и возникла парадигма структурного (процедурного) программирования

С программами, написанными в процедурном стиле, уже могли работать несколько человек, поддерживать их стало проще



Совсем потом

Но и на этом технические возможности не остановились - они продолжали расти. И вслед за ними росли сложность и объём новых программ. Возможности их ширились, но и требования к их разработке и обслуживанию тоже возрастали - нужно было больше людей, чтобы их писать и поддерживать

В рамках структурной парадигмы счёт функциям шёл на сотни и тысячи, управлять таким объёмом становилось тяжело, людям было тесно в структурной парадигме

И вот тогда в обиход вошла **объектно-ориентированная парадигма (ООП)**, которая ввела понятие **класса**

ООП

ООП позволило программистами отойти от необходимости представлять задачу в виде набора вызывающих друг друга функций. Вместо этого задача теперь представляется объектами, сообщающимися между собой. У объектов есть **поведение** - это их методы, и **состояние** (или **атрибуты**) - это значения их полей

Далее мы с вами рассмотрим **основные принципы** объектно-ориентированного программирования

Принципы ООП

это основополагающие механизмы,
которым должна следовать программа,
написанная в ОО парадигме

Принципы ООП

В ООП выделяют 4 основных принципа:

- Абстракция
- Инкапсуляция
- Наследование
- Полиморфизм

В этой и следующей лекции мы подробно рассмотрим каждый из них

Перерыв

Абстракция

2

Принцип абстракции

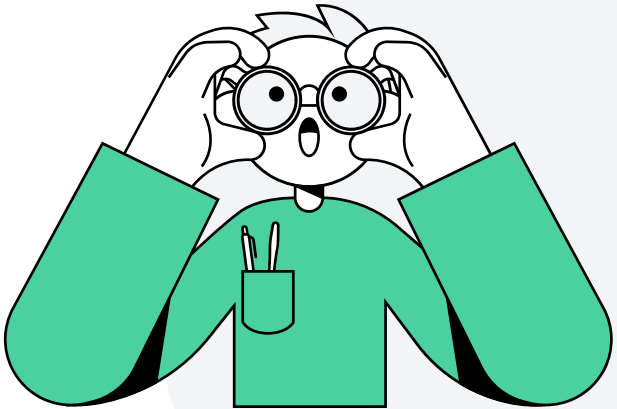
При проектировании классов нам нужно выделить из моделируемой сущности только те атрибуты и методы, которые имеют отношение к решению поставленной задачи

Пример

Представьте, что вы пишете программу, в которой вам предстоит взаимодействовать с телефоном. Вы понимаете, что вам нужно создать класс “Телефон”

Какими атрибутами и методами он должен обладать?

Для того, чтобы это выяснить, вы решили сесть на машину времени и слетать в 1970, 2000 и 2020 года - спросить у людей из этого времени, что такое телефон



Пример

Люди из 1970 года описали бы телефон так: это коробочка с барабаном, подключённая проводом. К коробочке проводом подключена трубка, которая на эту коробочку вешается

Чтобы позвонить, нужно снять трубку, набрать номер телефона на барабане - для этого надо крутить барабан на нужных цифрах - , приложить трубку к уху и ждать гудков

Чтобы ответить на звонок, нужно снять трубку с коробочки и приложить трубку к уху - можно разговаривать

Пример

Соответственно, вы выделяете следующие методы и атрибуты для класса “Телефон” из 1970 года

Атрибуты:

- Провод к телефону
- Сам телефонный аппарат
- Барабан
- Провод к трубке
- Трубка

Методы:

- Снять трубку
- Положить трубку
- Крутить барабан
- Набрать номер
- Ответить на звонок
- Приложить трубку к уху

Пример

Люди из 2000 года описали бы телефон так: это базовая станция, подключённая проводами. В базовой станции имеется отверстие, в которую вставляется трубка

Трубка имеет дисплей и кнопки. С трубкой можно ходить по всей квартире, но она может разрядиться. Чтобы зарядить трубку, нужно вставить её в базовую станцию. Трубку можно выключить и включить

Чтобы позвонить, нужно взять трубку, набрать номер телефона кнопками - для этого надо нажимать на нужные кнопки - , нажать кнопку вызова, приложить трубку к уху и ждать гудков

Чтобы ответить на звонок, нужно взять трубку, нажать кнопку ответа и приложить трубку к уху - можно разговаривать

Пример

Соответственно, вы выделяете следующие методы и атрибуты для класса “Телефон” из 2000 года

Атрибуты:

- Провода к базовой станции
- Базовая станция
- Трубка
- Кнопки на трубке
- Дисплей на трубке

Методы:

- Поставить трубку на базовую станцию
- Снять трубку с базовой станции
- Нажать на кнопку
- Набрать номер
- Ответить на звонок
- Приложить трубку к уху
- Ходить с трубкой по квартире
- Включить трубку
- Выключить трубку

Пример

Люди из 2020 года описали бы телефон так: это небольшое устройство (трубка), у которого есть сенсорный экран и две-три физических кнопки. В трубке есть отверстие для провода зарядки

Трубка может разрядиться, чтобы зарядить её, нужно воткнуть провод с питанием в отверстие или положить на специальную площадку, подключенную проводом. С трубкой можно ходить везде, где есть сотовая связь. Трубку можно включить и выключить

Чтобы позвонить, нужно взять трубку, открыть приложение звонков, набрать номер телефона кнопками - для этого надо нажимать на нужные кнопки - , нажать кнопку вызова, приложить трубку к уху и ждать гудков

Чтобы ответить на звонок, нужно взять трубку, нажать кнопку ответа и приложить трубку к уху - можно разговаривать

Пример

Соответственно, вы выделяете следующие методы и атрибуты для класса “Телефон” из 2020 года

Атрибуты:

- Провод для зарядки
- Трубка
- Кнопки на трубке
- Сенсорный экран

Методы:

- Поставить трубку на зарядку
- Снять трубку с зарядки
- Нажать на кнопку
- Набрать номер
- Ответить на звонок
- Приложить трубку к уху
- Ходить с трубкой везде
- Включить трубку
- Выключить трубку

Пример

Как мы видим, в зависимости от того, из какой временной точки мы описываем одно и то же понятие, получается совсем разный результат. А теперь представьте, что вы попросили бы описать современный смартфон маленького мальчика, инженера-электротехника и маркетолога - получили бы тоже совсем разные описания!

Тем не менее, когда мы будем разрабатывать класс, нам нужно будет выбрать конкретные атрибуты и конкретные методы, которые будет реализовывать наш класс. Как вы уже убедились, можно сделать это совершенно по-разному. И решить, как это сделать **лучше**, нам помогает принцип абстракции

Пример

Принцип абстракции говорит нам - возьмите только то поведение и свойства, которые нужны для решения поставленной задачи. Соответственно, первое, что нам нужно сделать - **чётко сформулировать задачу, которую должен решить наш класс**

Например, в рамках нашей программы нам нужно звонить по телефону и отвечать на звонки - не более

Пример

И теперь, основываясь на поставленной задаче и проведённом анализе, мы можем выделить нужные нам атрибуты и методы:

Атрибуты:

- Трубка

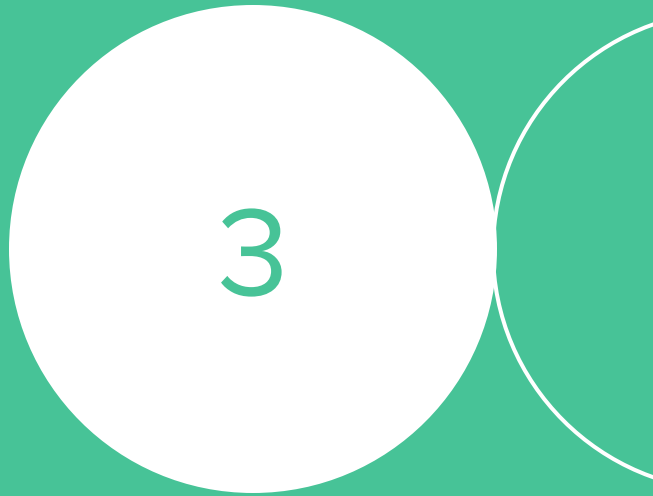
Методы:

- Набрать номер
- Ответить на звонок

Итог

Чем меньшим количеством кода вы обойдётесь в классе для решения поставленной задачи - тем лучше, потому что в будущем будет легче в нём разобраться и изменить. Принцип абстракции помогает нам **абстрагироваться** от ненужных деталей моделируемого объекта

Инкапсуляция



Инкапсуляция

с англ. Encapsulation - образовано
от словосочетания **in capsulo**,
что означает “в оболочке”

Принцип инкапсуляции

Любой класс должен рассматриваться как “чёрный ящик” - то есть класс должен предоставлять наружу строго определённые методы и/или атрибуты, которых достаточно внешним пользователям для того, чтобы пользоваться возможностями класса

При этом класс должен выполнять возложенные на него задачи внутри себя (в том числе с использованием других классов), но внешний пользователь класса не должен знать о том, как класс выполняет свои задачи

Зачем это нужно

Соблюдение принципа инкапсуляции делает наши классы более стабильными - они защищены от неожиданного внешнего воздействия на своё состояние

Также классы становятся более удобными в использовании - вам не нужно знать детали реализации класса, чтобы им воспользоваться (вспомните класс std::ifstream - чтобы закрыть файл, вам нужно вызвать функцию *close*, но при этом вы понятия не имеете, как именно он закрывает файл)

А ещё повышается независимость класса - его легче использовать в других программах

Как добиться инкапсуляции

Как вы думаете, что нам помогает соблюдать принцип инкапсуляции?

Как добиться инкапсуляции

В C++ соблюдать принцип инкапсуляции нам помогают модификаторы доступа, которые мы изучили на прошлой лекции

В частности, модификаторы доступа private и protected скрывают помеченные ими члены класса от внешних пользователей - тем самым гарантируя, что извне они не будут вызваны. Эти модификаторы доступа помогают нам скрыть внутренние методы и поля, которые нужны классу для работы

В то же самое время, модификатор доступа public определяет, какие методы и поля будут доступны внешнему пользователю. И это тоже важно для соблюдения принципа инкапсуляции - чётко определить члены класса, предназначенные для взаимодействия с внешним пользователем класса

Итоги

Итоги занятия

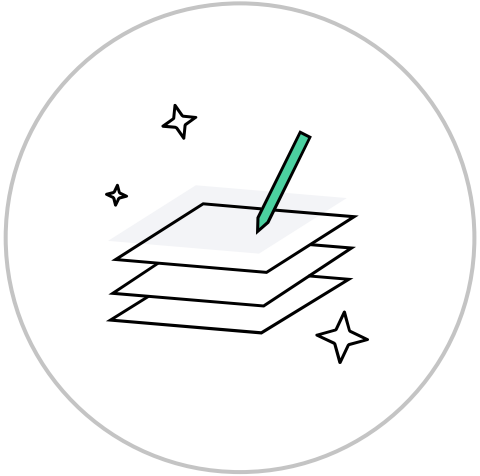
Сегодня мы

1. Разобрались, что такое ООП
2. Познакомились с историей ООП
3. Узнали, что такое абстракция
4. Выяснили, что такое инкапсуляция

Домашнее задание

Давайте посмотрим ваше домашнее задание.

1. Вопросы по домашней работе задавайте в чате группы
2. Задачи можно сдавать по частям
3. Зачёт по домашней работе ставят после того, как приняты все задачи



Дополнительные материалы

- Абстракция
- Инкапсуляция

Задавайте вопросы и пишите отзыв о лекции

Максим Бакиров
C++ - разработчик в Яндекс