Используйте ресурсы

[ООП в картинках](https://habr.com/ru/post/463125/) (Habr)

Google/Yandex

Читаем, отвечаем на вопросы.

**часть1. Читаем до первого примера кода. Ответы вставляем после вопроса**

**1.** Опишите известные вам парадигмы программирования.

Парадигмы программирования — это фундаментальные стили или подходы к созданию программного обеспечения. Они обеспечивают различные способы концептуализации и структурирования программ. Вот некоторые из наиболее распространённых:

* **Императивное программирование:**
  + **Концепция:** фокусируется на описании *того, как* работает программа, путем указания последовательности команд. Это похоже на пошаговые инструкции для компьютера.
  + **Примеры:** процедурное программирование, структурное программирование. Такие языки, как C, Pascal и Fortran, часто ассоциируются с императивными парадигмами.
  + **Характеристики:** используются переменные, присваивания и структуры управления (циклы, условные операторы).
* **Декларативное программирование:**
  + **Концепция:** фокусируется на описании *того, что* должна делать программа, без указания точных шагов, как это сделать. Она выражает логику вычислений без описания потока управления.
  + **Примеры:** функциональное программирование, логическое программирование, языки запросов к базам данных (например, SQL). Такие языки, как Haskell, Lisp и Prolog, связаны с декларативным программированием.
  + **Характеристики:** опирается на функции, логические выражения и неизменяемые данные.
* **Процедурное программирование:**
  + **Концепция:** Подмножество императивного программирования, в котором программы разбиваются на процедуры или функции.
  + **Примеры:** C, Pascal, Fortran.
  + **Характеристики:** Подчеркивает модульность за счет процедур, которые являются многократно используемыми блоками кода.
* **Объектно-ориентированное программирование (ООП):**
  + **Концепция:** фокусируется на моделировании объектов реального мира как «объектов», обладающих данными (атрибутами) и поведением (методами).
  + **Примеры:** Java, C++, Python, C#, Smalltalk.
  + **Характеристики:** инкапсуляция, наследование, полиморфизм, абстракция и организация на основе классов.
* **Функциональное программирование:**
  + **Концепция:** рассматривает вычисления как вычисление математических функций и позволяет избежать изменения состояния и изменяемых данных.
  + **Примеры:** Haskell, Lisp, Clojure.
  + **Характеристики:** использует неизменяемые данные, функции как объекты первого класса, рекурсию, чистые функции и лямбда-выражения.
* **Логическое программирование:**
  + **Концепция:** использует логические правила и факты для выражения логики программы.
  + **Примеры:** Пролог.
  + **Характеристики:** использует логический вывод и возврат к предыдущим результатам.

**2.** При разработке проекта на ООП – проекцию чего реализует подход ООП**?**

В ООП мы моделируем реальные проблемы с помощью объектов. Таким образом, подход ООП реализует проекцию **реальной проблемной области** на программную систему. ООП пытается отразить объекты реального мира и их взаимодействие в коде, чтобы мы могли моделировать реальный мир в нашем коде. Это делает код более понятным и простым в изменении и поддержке.

**3.** Когда реализация программного проекта на ООП приносит максимальную выгоду**?**

ООП приносит максимальную пользу, когда:

* **Работа со сложностью:** когда в проектах есть большое количество взаимосвязанных компонентов или сложных взаимодействий. ООП предоставляет метод управления этой сложностью за счет разделения системы на логические единицы (объекты).
* **Компоненты многократного использования:** когда вам нужно повторно использовать компоненты или расширять их с помощью наследования.
* **Большие и постоянно развивающиеся кодовые базы:** когда вам нужно со временем добавлять новые функции. ООП делает код более удобным для сопровождения и расширения.
* **Моделирование объектов реального мира:** когда предметная область состоит из объектов реального мира с их данными и поведением.
* **Командные проекты:** ООП способствует модульности и инкапсуляции, позволяя командам работать над разными частями одного проекта без серьёзных конфликтов.
* **Когда вам нужны чёткие и понятные модели:** ООП очень хорошо подходит для создания модели, которую можно обсуждать с людьми, не имеющими глубоких знаний в области компьютерных наук.

**4.** Когда вполне можно обойтись без ООП**?**

Часто можно обходить ООП, когда:

* **Простые программы:** для небольших, простых программ с минимальной логикой. ООП может добавить ненужные издержки и усложнить работу.
* **Скрипты:** для быстрых одноразовых скриптов или прототипов, где процедурный подход проще и быстрее.
* **Приложения, критичные к производительности:** в некоторых сценариях издержки ООП могут повлиять на производительность. Если максимальная производительность имеет решающее значение, более подходящим может оказаться императивный/процедурный подход, хотя в современных средах ООП разница в производительности уменьшилась.
* **Обработка данных:** в некоторых сценариях обработки данных лучше подходят функциональные парадигмы или парадигмы потока данных, так как они хорошо справляются с преобразованием данных.

**5.** ООП ускоряет ли создание программного продукта**?**

Ответ зависит от обстоятельств. ООП может привести к увеличению скорости в *долгосрочной перспективе*, но в краткосрочной перспективе может быть медленнее:

* **Краткосрочная перспектива:** первоначальный дизайн ООП-программы может занять больше времени, чем создание аналогичной процедурной программы, поскольку вам нужно смоделировать объекты, их свойства и поведение.
* **Долгосрочная перспектива:** ООП помогает:
  + **Сокращение времени разработки:** благодаря многократному использованию компонентов (классов и т. д.) вы можете сократить время разработки.
  + **Сокращение времени исправления ошибок:** ООП обеспечивает четкую структуру и инкапсуляцию, снижая риск возникновения ошибок при работе с одним компонентом, а также помогает легче локализовать ошибки.
  + **Улучшенная поддержка:** инкапсуляция упрощает поддержку проекта и внесение изменений, необходимых для будущих версий и обновлений

**6.** Поясните понятие «класс», приведите пример класса из окружающего мира (он нам потребуется ниже). Рассмотрите известную вам предметную область, например, автомобили, услуги салона красоты.

**Класс** — это схема или шаблон для создания объектов. Он определяет атрибуты (данные) и поведение (методы), которые будут у объектов этого класса. Это не сам объект, а рецепт для создания объектов.

Давайте воспользуемся примером класса **“Car”**:

* **Класс:** Car (Это чертёж).
* **Объекты:** Каждый отдельный автомобиль является *объектом* класса Car (например, мой автомобиль — это объект, ваш автомобиль — это объект). Каждый объект будет иметь собственное состояние (например, цвет), но будет обладать одинаковыми свойствами (например, у него есть скорость, и на нём можно ездить).
* **Атрибуты (поля):**
  + modelName (Строка)
  + color (Строка)
  + speed (int)
  + numberOfDoors (int)
* **Методы (Поведение):**
  + start()
  + accelerate()
  + brake()
  + turnLeft()
  + turnRight()

**7.** Что такое «поля класса»?

**Поля класса**, также называемые *переменными экземпляра* или *атрибутами*, представляют собой данные, связанные с каждым *объектом* (экземпляром) класса. Они определяют состояние объекта. В нашем Car примере выше, modelName, color, speed и numberOfDoors - это все поля. Каждый объект car, созданный из класса Car, будет иметь свои собственные значения для каждого из этих полей.

**8.** Что такое «методы класса»?

**Методы класса** - это операции или действия, которые может выполнять объект. Они представляют поведение объекта. В нашем Car примере, start(), accelerate(), brake() turnLeft(), turnRight() и все это методы. Эти методы определяют, что может *делать* автомобиль.

**9.** На своем примере класса укажите поля и методы класса.

Используя класс «Автомобиль» из примера выше:

* **Поля:**
  + modelName (Строка) — например, «Toyota Camry»
  + color (Строка) — например, «Красный»
  + speed (Целое число) — например, 60 (км/ч)
  + numberOfDoors (Целое число) — например, 4
* **Методы:**
  + start() - Запускает двигатель автомобиля.
  + accelerate(int increaseSpeed) - Увеличивает скорость автомобиля. (Мы можем сделать наши методы более полезными, используя параметры.)
  + brake() - Замедляет или останавливает автомобиль.
  + turnLeft() - Поворачивает автомобиль налево.
  + turnRight() - Поворачивает автомобиль направо.