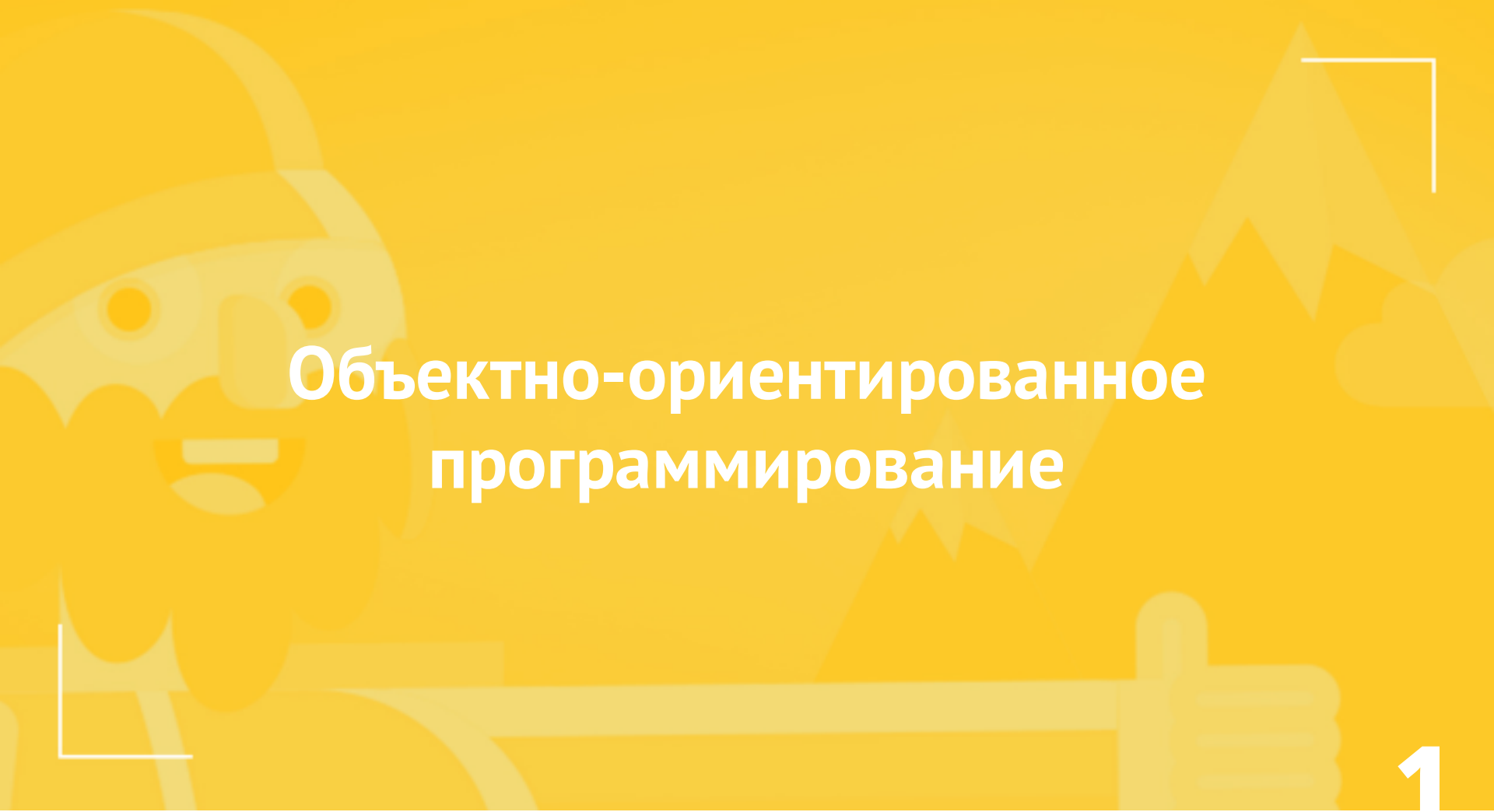


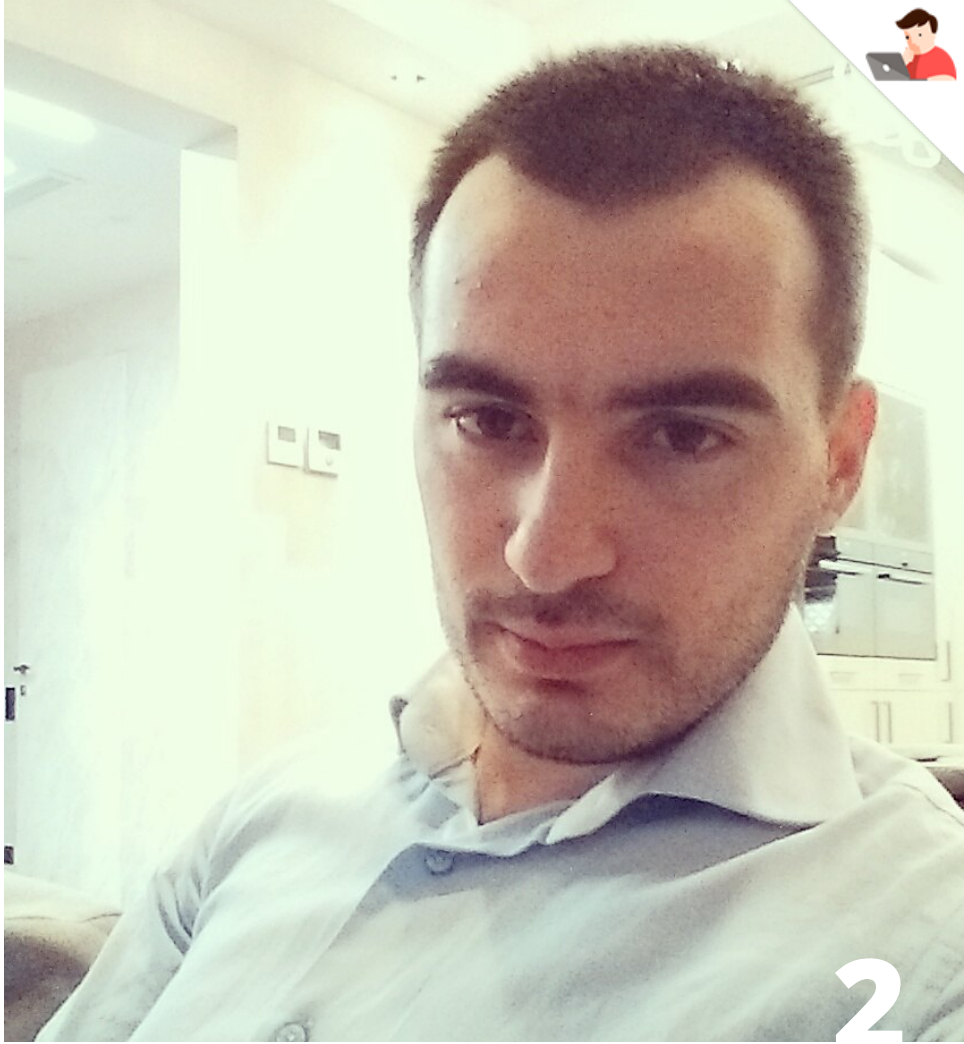
Объектно-ориентированное программирование

Ведущий вебинара

Сергей Мелюков

Круг интересов: Backend, Frontend, GameDev, MobileDev

Место работы: Frontend разработчик профессиональных инструментов Avito



Содержание

Инкапсуляция

Инкапсуляция

Предположим: необходимо воспроизвести видео-файл с внешнего сервера.

Для этого нам потребуется отправить запрос на внешний сервер, получить файл, разместить его на странице и запустить воспроизведение.

Но вместо этого, мы можем создать один метод **play**(), код которого сам вызовет все указанные действия в нужном порядке.

Таким образом мы скроем более сложные задачи (отправка запрос, получение файл и воспроизведение), за простым методом **play**()

Инкапсуляция - парадигма ООП, которая утверждает, что нужно скрывать реализацию более сложных вещей за более простыми.

Паттерн “модуль”

Паттерн “модуль”

Паттерн - **проверенный** временем способ решения **конкретной** задача.

Паттерн “модуль” - способ **организации** кода при котором одна часть кода **скрыта** от глаз конечного пользователя, а другая его часть **открыта** и позволяет **управлять** скрытой частью.

Чаще всего используется для **разделения** кода на логические части.

Как вариант инкапсуляции в JS

Наследование

Наследование

Предположим: мы создаем программу для рисования и уже реализовали класс, который рисует круг.

Далее, нам понадобилось добавить еще одну фигуры - круг, но который можно заливать определенным цветом.

Это значит, что нужно создать новый класс, который унаследует все возможности класса, который умеет рисовать простой круг, и добавить к этому новому классу, новые возможности - заливка круга цветом.

При этом, поведение старого класса не изменится.

Полиморфизм

Полиморфизм

Предположим: есть задача - отобразить на экране мультимедиа-файл (картинку, аудио или видео). Для картинки - вывод на экран. Для видео - воспроизведение видео ряда. Для аудио - воспроизведение звука.

Классы для картинки, видео и аудио имеет разную реализацию воспроизведения, но если они поддерживают спецификацию “воспроизведение информации” (имеют метод **play**()), то мы можем даже не знать - какой тип данных мы пытаемся воспроизвести, а просто вызвать метод **play**() у переданного объекта. И в зависимости от того, что именно было передано (картинка, видео или аудио) - будет выполнен соответствующий код, отвечающий за воспроизведение.

Это и есть суть полиморфизма - одна спецификация может иметь множество вариантов реализации и можно не заботиться о том, с каким из вариантов реализации мы сейчас работаем.

Конструкторы

Конструкторы

Если вызвать функцию с ключевым словом **new**, то функция начнет возвращать “пустой” объект.

Такая функция называется **конструктором**, а объект, которая она возвращает, **экземпляром**.

this, внутри этой функции, будет указывать на *экземпляр*.

Внутри экземпляра всегда есть свойство **constructor**, которое указывает на функцию-конструктор.

Прототипное наследование

Прототипное наследование

У каждого конструктора есть свойство **prototype**, которое изначально является “пустым” объектом.

Наполняя **prototype** свойствами, мы, тем самым, наполняем теми же свойствами и **все экземпляры** данного конструктора.

Когда конструктор создает экземпляр, помимо свойства **constructor**, в экземпляр помещается **скрытое** свойство **__proto__**, которое указывает на **prototype** конструктора.

Прямой доступ к **__proto__** доступен *не во всех браузерах*.

Прототипное наследование

Когда мы пытаемся обратиться к какому либо свойству экземпляра:

- происходит поиск свойства в экземпляре
- если не найдено - искать в служебном свойстве **__proto__**
- предыдущий пункт повторяется пока:
    - либо не найдется свойство
    - либо не закончится вложенность

Кстати, свойство **constructor** располагается в **prototype** конструктора, а не добавляется в экземпляр при его создании.

Прототипное наследование

Когда При необходимости реализовать прототипное наследование, **не рекомендуется** присваивать один прототип другому.

При присваивании объектов, оба объекта начинают указывать **на один и тот же объект**, следовательно, изменяя один из объектов, автоматически изменяется и другой.

Для реализации правильного прототипного наследования, необходимо:

- либо создать объект-прослойку между прототипами
- либо использовать **Object.create**()

Время ваших вопросов