

# Объектно-ориентированное программирование

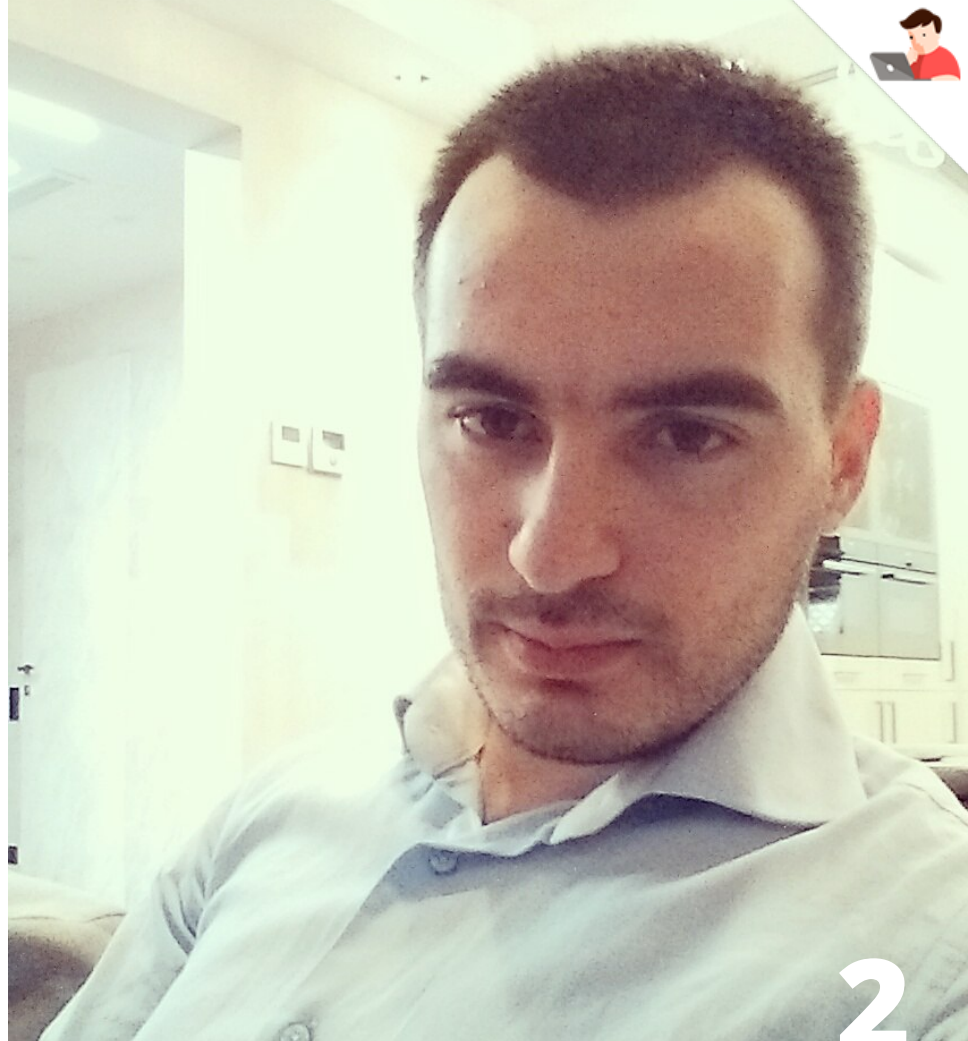
# Ведущий вебинара

---

**Сергей Мелюков**

**Круг интересов:** Backend, Frontend, GameDev, MobileDev

**Место работы:** Frontend разработчик профессиональных инструментов Avito



# Содержание

---



1. Инкапсуляция
2. Паттерн “модуль”
3. Наследование
4. Полиморфизм
5. Конструкторы
6. Прототипное наследование

The background features a stylized illustration of a mountain range with several peaks in shades of gray. A white, fluffy cloud is positioned on the left side, partially obscuring one of the mountain peaks. In the top right corner, there is a yellow L-shaped graphic element. In the bottom left corner, there is another yellow L-shaped graphic element. The word "Инкапсуляция" is centered in the middle of the image in a bold, black, sans-serif font.

# 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.

The background features a stylized illustration of mountains in shades of gray and white, with a single white cloud on the left side. There are also yellow L-shaped decorative elements in the top right and bottom left corners.

# 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

The background features a minimalist illustration of a mountain range with several peaks in shades of gray. A single white cloud is positioned on the left side. The entire scene is set against a light gray background. In the top right and bottom left corners, there are yellow L-shaped decorative lines. A solid yellow horizontal bar runs along the bottom edge of the slide.

# 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.

The background features a stylized illustration of mountains in shades of gray and white, with a white cloud on the left. Yellow L-shaped corner brackets are positioned in the top right and bottom left corners.

# 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 имеют 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.

The background features a stylized illustration of a mountain range with several peaks in shades of gray. A white cloud is positioned on the left side, partially overlapping the mountains. In the top right corner, there is a yellow L-shaped line. In the bottom left corner, there is another yellow L-shaped line. The word "Конструкторы" is centered in the middle of the slide in a bold, black, sans-serif font.

# Конструкторы



# Конструкторы

---

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

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

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

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

The background features a light gray sky with a white cloud on the left and several stylized mountain peaks in shades of gray. In the bottom right corner, there is a large yellow number '14'. There are also yellow L-shaped decorative elements in the top right and bottom left corners.

# **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**