Одиночка

Singleton Pattern

«порождающий шаблон проектирования,

гарантирующий единственность экземпляра некоторого класса,
и
предоставляющий глобальную точку доступа к этому экземпляру»

почти **википедия**

когда использовать

- нужен легкодоступный единственный экземпляр
- расширения экземпляра не должны влиять на внешний код

самое частое использование - работа с внешними интерфейсами

реализация

```
class FileSystem {
  constructor() {
    if (FileSystem.instance)
      return FileSystem.instance;

    FileSystem.instance = this;
  }
}
```

что есть хорошего?

- Единственный экземпляр класса
- Глобальный доступ к этому экземпляру
- «Ленивая» инициализация

пример

```
class WindowsFileSystem extends FileSystem {
  readFile(path) { ... }
  writeFile(path, data) { ... }
}

class LinuxFileSystem extends FileSystem {
  readFile(path) { ... }
  writeFile(path, data) { ... }
}

FileSystem.instance = platform === 'windows'
  ? new WindowsFileSystem()
  : new LinuxFileSystem();
```

что есть плохого?

- Единственный экземпляр
- Глобальный доступ к экземпляру
    - Менее понятный код
    - Высокая «связность» кода
- Он решает две проблемы, даже если у вас всего одна
- «Ленивая» инициализация

улучшения?

- Нужен ли вам этот класс вообще?
    - Дважды подумайте, если в названии есть **Manager** или **Helper**
- Нужен ли вам глобальный доступ?
    - Приватность объекта?
    - Удобство доступа?
- Инициализация при запуске программы?

сахарозаменители

- иногда вам нужен просто **статический** класс
    - или просто **статический флаг**
- **прямая передача** объекта
    - **dependency injection** и похожие
- **получение** объекта **из базового класса**
- **получение из** другого **глобального объекта**
- А еще есть ***другие паттерны*** ⟶

домашнее задание

- **Object Pool**
- **Service Locator**
- **Sandbox Subclass**
- **Factory Method**
- **Strategy**

Aspect-Oriented programming

Спасибо за внимание!

ModulBank, 2018