

Структурные шаблоны проектирования

Иван Поляков

Разработчик Go/C++ в инфраструктуре поиска в Авито



Проверка связи



Если у вас нет звука:

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



Поставьте в чат:

-  если меня видно и слышно
-  если нет

Иван Поляков

О спикере:

- Разработчик Go/C++ в инфраструктуре поиска в Авито
- 5 лет работал в Nexign, писал real time сервисы для телекома Мегафона на C++



Вспоминаем прошрое занятие

Вопрос: что такое паттерн проектирования?



Вспоминаем прошрое занятие

Вопрос: что такое паттерн проектирования?

Ответ: типичный способ решения часто встречающихся проблем/задач при проектировании программ



Вспоминаем прошрое занятие

Вопрос: зачем нужен паттерн «Фабричный метод»?



Вспоминаем прошрое занятие

Вопрос: зачем нужен паттерн «Фабричный метод»?

Ответ:

- описать интерфейс создания объектов в родительском классе
- делегировать создание конкретных реализаций в классах-наследниках



Вспоминаем прошрое занятие

Вопрос: зачем нужен паттерн «Абстрактная фабрика»?



Вспоминаем прошрое занятие

Вопрос: зачем нужен паттерн «Абстрактная фабрика»?

Ответ:

- расширение идеи «Фабричного метода»
- предоставляет интерфейс для создания семейств связанных объектов



Вспоминаем прошрое занятие

Вопрос: зачем нужен паттерн «Строитель»?



Вспоминаем прошрое занятие

Вопрос: зачем нужен паттерн «Строитель»?

Ответ: делегировать создание сложных объектов в отдельный класс



Вспоминаем прошрое занятие

Вопрос: зачем нужен паттерн «Одиночка»?



Вспоминаем прошрое занятие

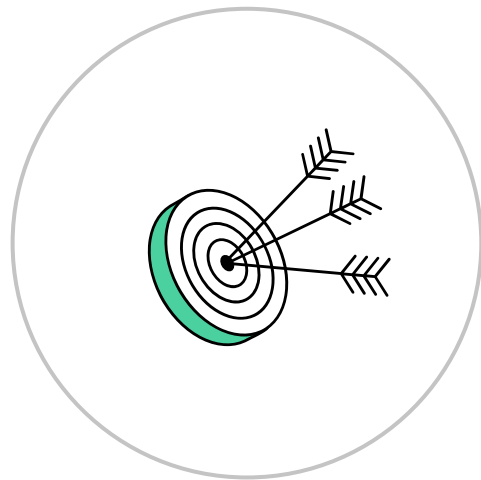
Вопрос: зачем нужен паттерн «Одиночка»?

Ответ: контролировать доступ к объекту из
всех частей программы



Цели занятия

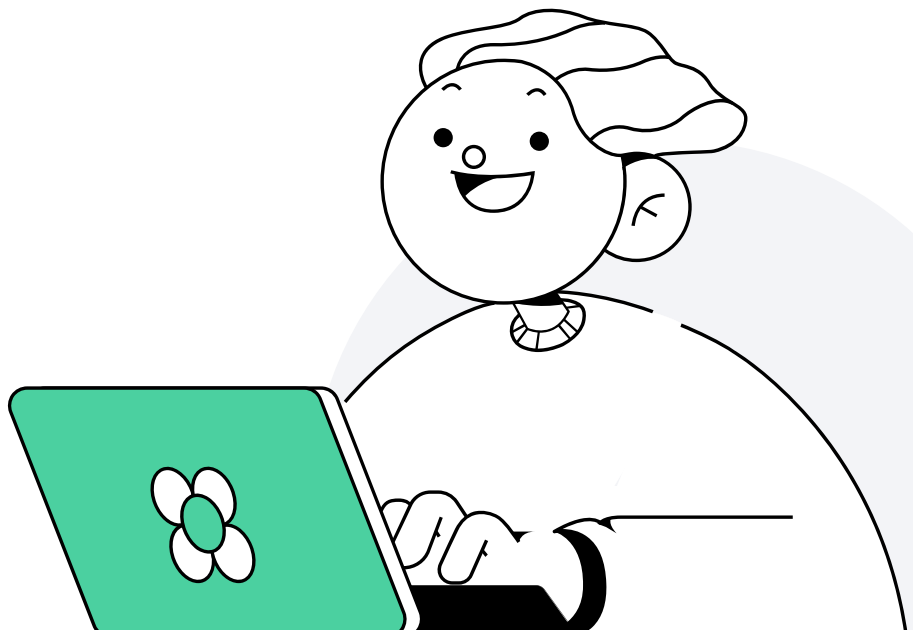
- Познакомиться со структурными паттернами проектирования
- Научиться разрабатывать программы с использованием структурных паттернов



План занятия

- 1 Структурные паттерны
- 2 Адаптер
- 3 Декоратор
- 4 Заместитель
- 5 Итоги
- 6 Домашнее задание

*Нажми на нужный раздел для перехода



Структурные паттерны

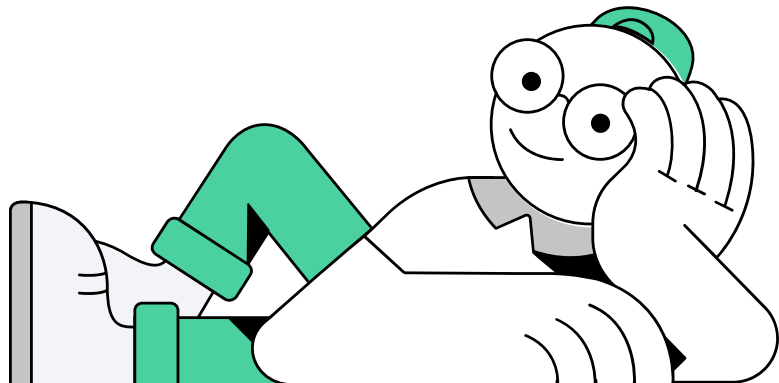


1

Структурные паттерны

Описывают построение иерархий классов:

- Слабосвязанных
- Легких в поддержке



Структурные паттерны

1. Адаптер
2. Декоратор
3. Прокси (Заместитель)
4. Мост
5. Фасад
6. Компоновщик

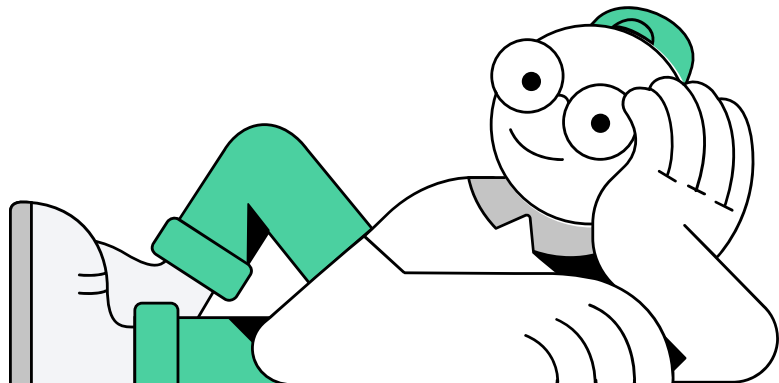
Адаптер



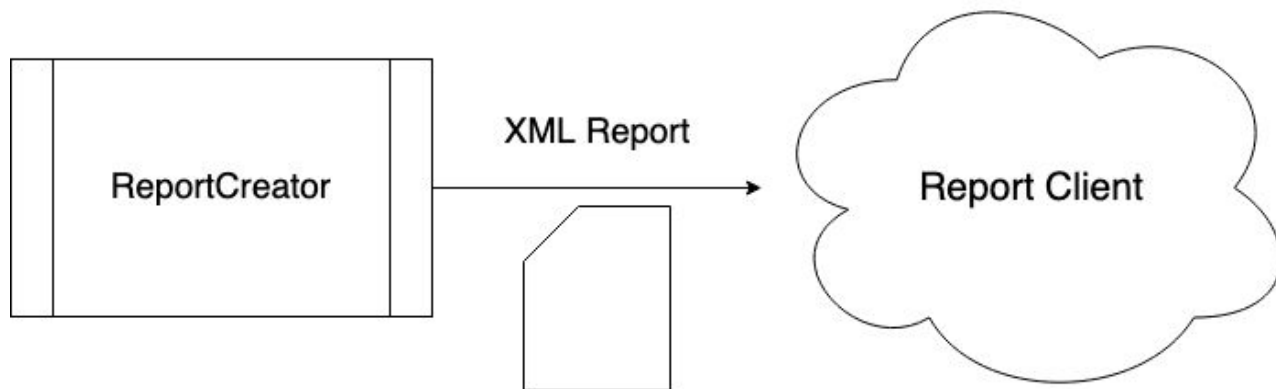
2

Адаптер

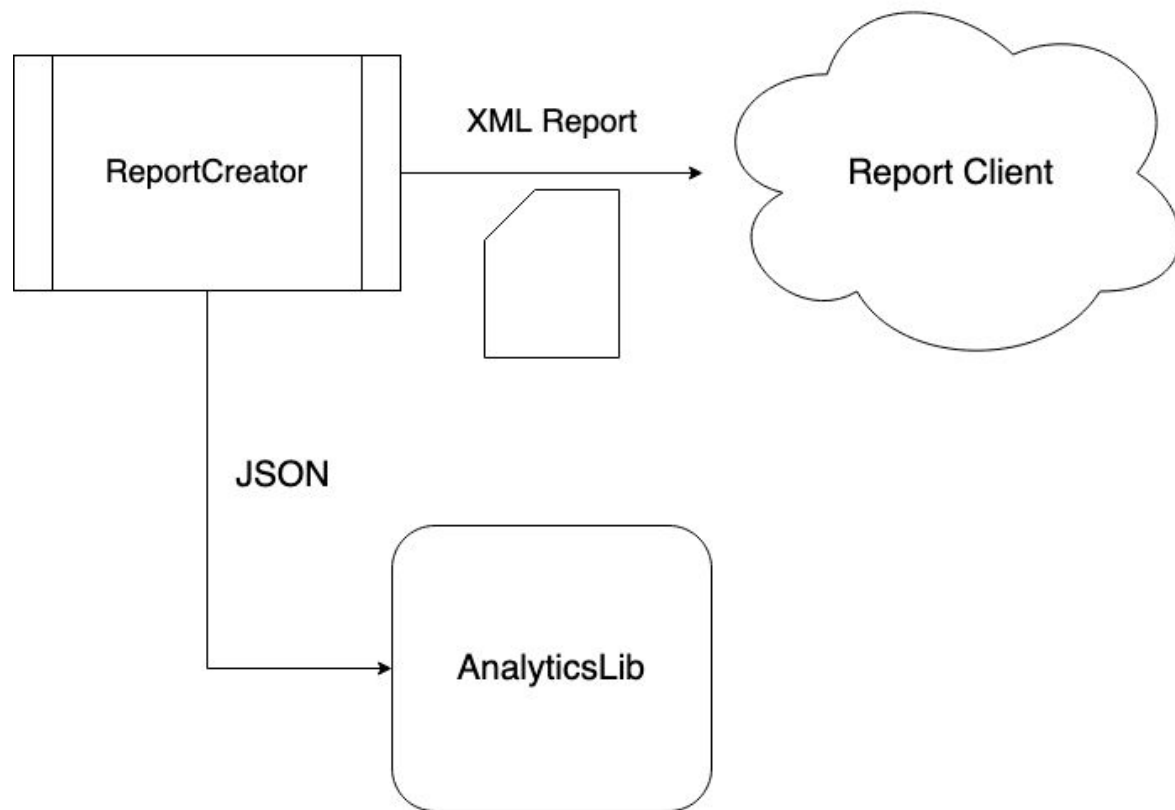
Позволяет использовать вместе объекты с несовместимым интерфейсом



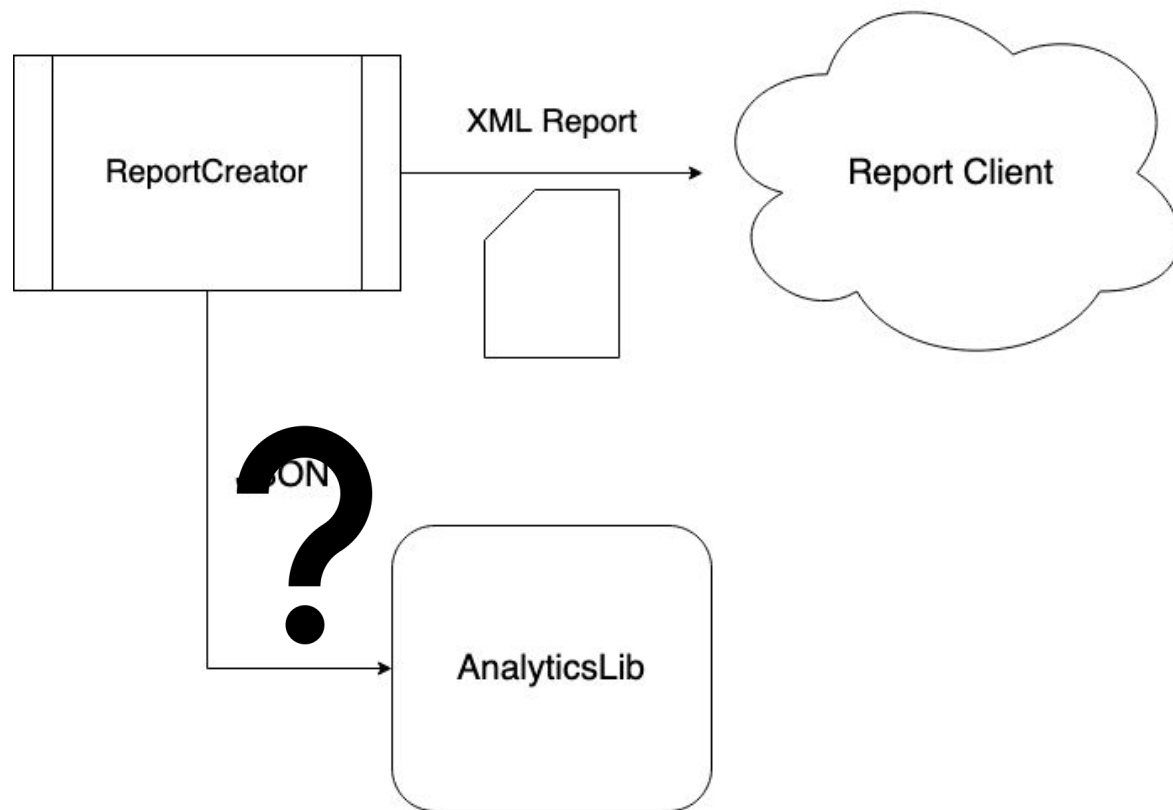
Адаптер (пример)



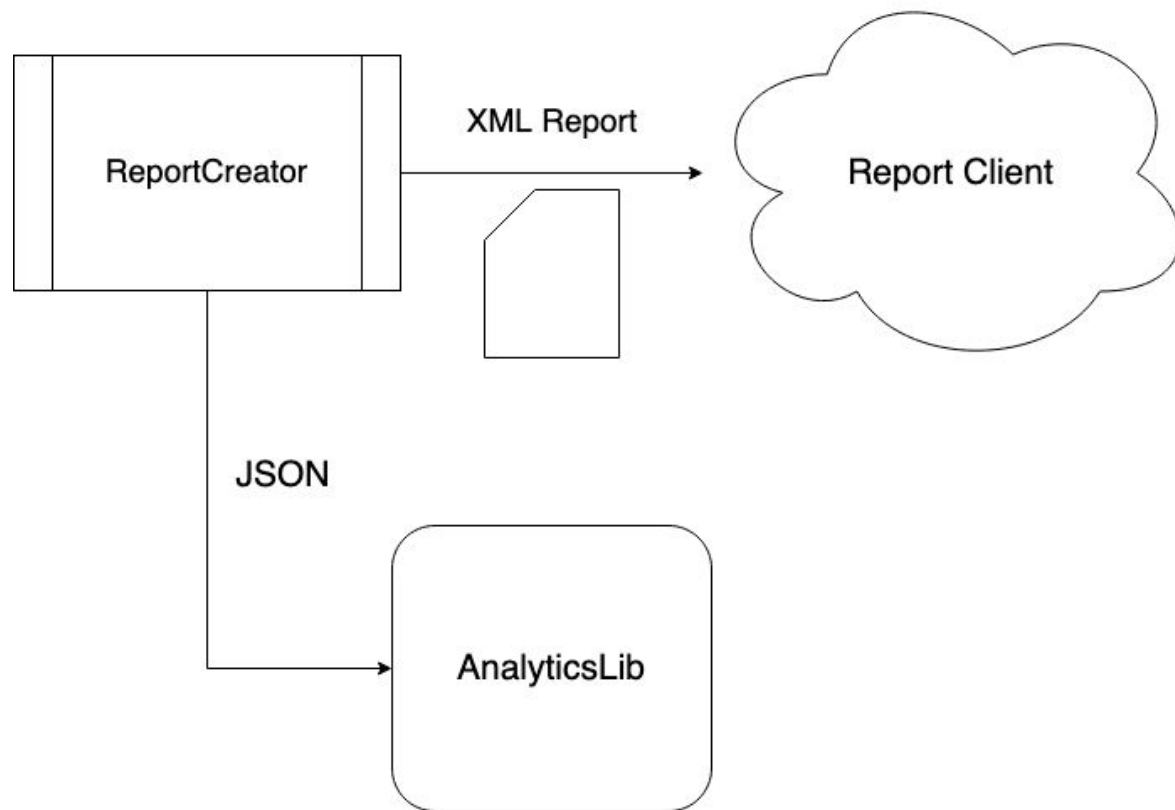
Адаптер (пример)



Адаптер (пример)



Адаптер (пример)



Декоратор

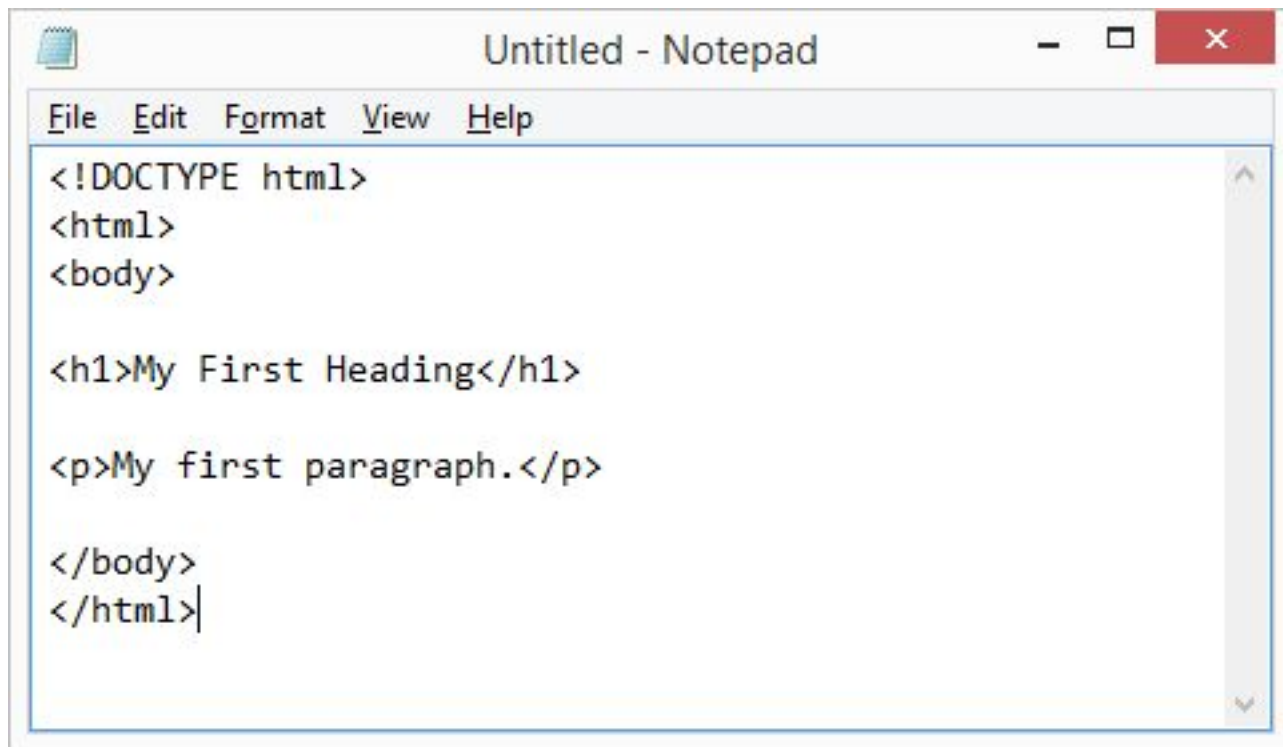
3

A decorative graphic in the bottom right corner consisting of two overlapping circles. The left circle is white with a dark teal outline and contains the number '3' in dark teal. The right circle is dark teal with a white outline and is partially cut off by the edge of the frame.

Декоратор

Добавляет поведение объектам, не меняя интерфейс

Декоратор (пример)



A screenshot of a Notepad window titled "Untitled - Notepad". The window has a standard menu bar with "File", "Edit", "Format", "View", and "Help". The text area contains the following HTML code:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Heading</h1>  
  
<p>My first paragraph.</p>  
  
</body>  
</html>|
```

The cursor is positioned at the end of the last line of code. The window has a scrollbar on the right side.

Декоратор (и Адаптер)

- **Адаптер** **меняет** интерфейс класса, добавляя поведение
- **Декоратор** **не меняет** интерфейс, добавляя поведение

Декоратор (примеры использования)

1. Замер времени выполнения участка кода
2. Дополнительное логирование
3. Проверка доступа
4. Насыщение параметрами контекста для передачи дальше (или удаление лишнего)

Прокси (Заместитель)



4

Заместитель

Перехватывает вызовы к реальному объекту, добавляя свою логику

Заместитель (примеры использования)

1. Кэширование
2. “Ленивая” инициализация
3. Настройка доступа
4. Тестирование

Заместитель и Декоратор

- **Заместитель** управляет жизненным циклом реального объекта
- **Декоратор** может и не владеть декорируемым объектом

Все-все-все

- **Адаптер** **меняет** интерфейс класса, добавляя поведение
- **Декоратор** **не меняет** интерфейс, добавляя поведение (расширяет интерфейс)
- **Заместитель** повторяет интерфейс замещаемого объекта

Итоги занятия

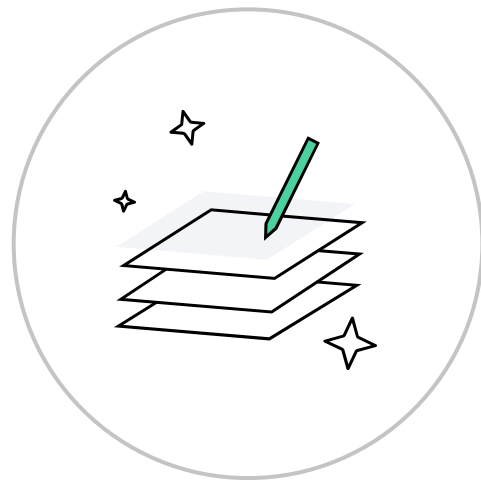
- 1 Повторили порождающие паттерны
- 2 Познакомились с основными структурными паттернами
- 3 Разобрали паттерны Адаптер, Заместитель и Декоратор



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



Задавайте вопросы и пишите отзыв о лекции

Иван Поляков

Разработчик Go/C++ в инфраструктуре поиска в Авито

