Структурные паттерны

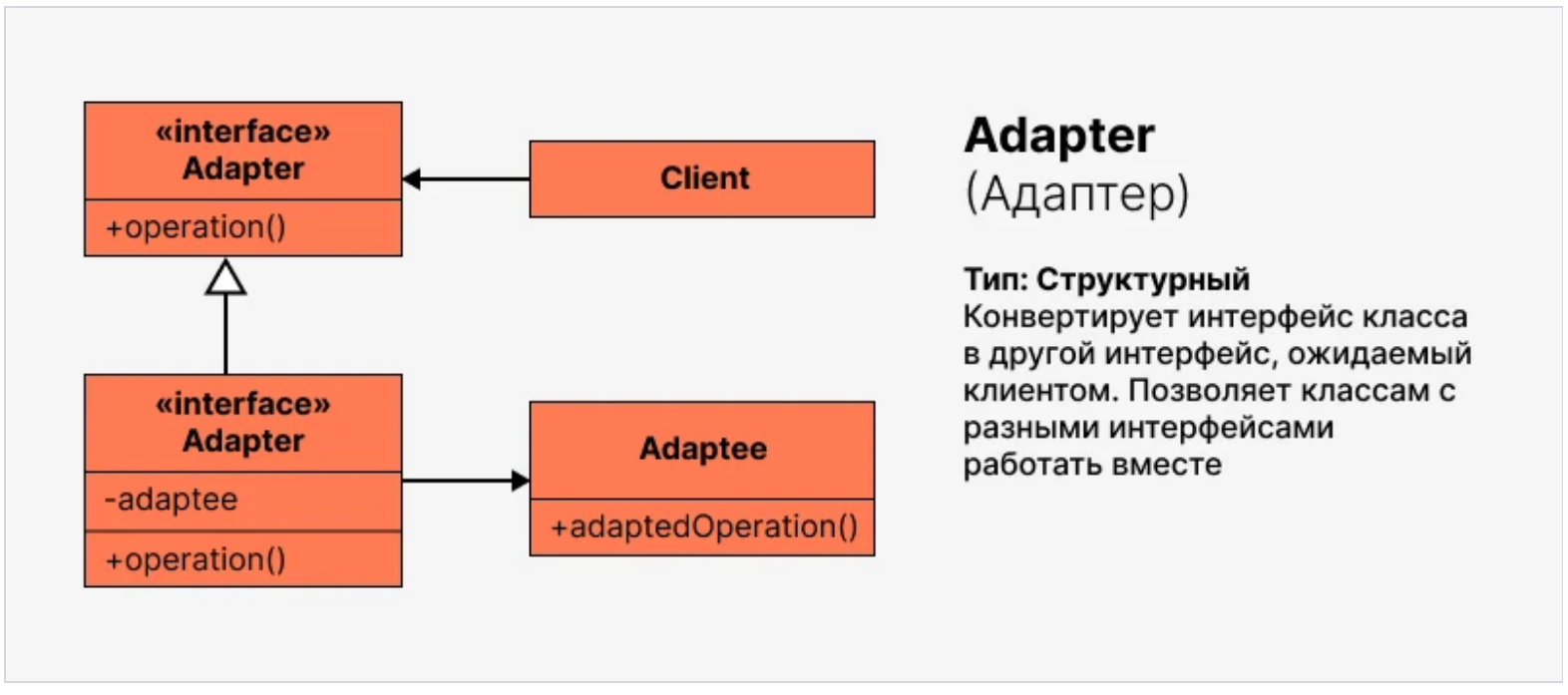
JSP & Servlets
16 уровень, 1 лекция

ОТКРЫТА

2.1 Adapter

Адаптер (Adapter) — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Официальное определение немного сложновато воспринимается, но если описать его своими словами, то адаптер — это паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.



Используется для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс. Создается дополнительный класс, у которого есть нужный интерфейс, а этот класс уже в свою очередь вызывает методы нужного объекта (у которого нет требуемого интерфейса).

Важно! Если в коде ты встречаешь у класса суффикс Adapter, то имеешь полное право считать, что этот класс выполняет роль адаптера и связан с группой классов, которые работают по описанной выше схеме.

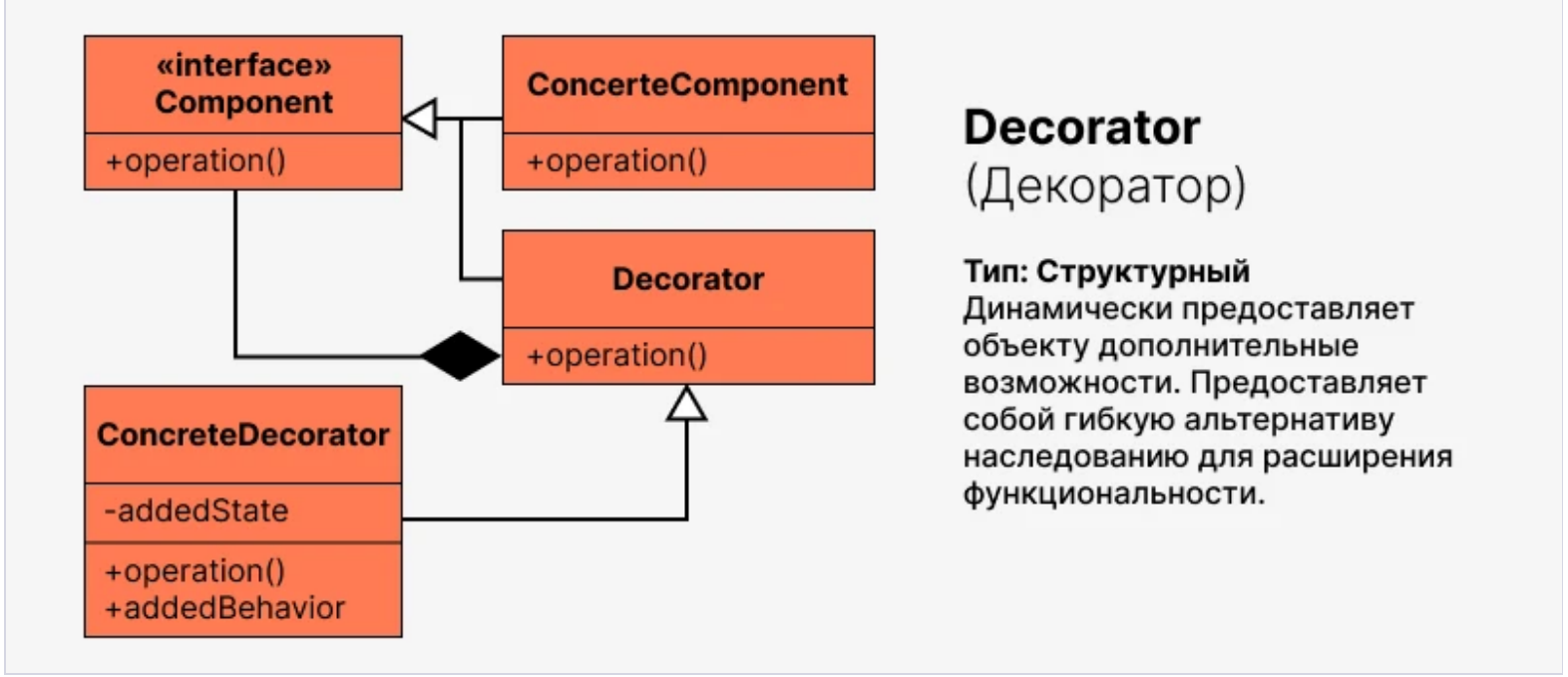
Применяется в случаях, когда система поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс. Чаще всего шаблон Адаптер применяется, если необходимо создать класс, унаследованный от нового или уже существующего абстрактного класса.

Сильные стороны:

- Переход на использование других внешних классов не требует переделки самой системы, достаточно реализовать еще один класс Adapter.
- Независимость от реализации внешних классов (классов из библиотек, чей код мы не можем поменять). Твоя программа становится независимой от интерфейса внешних классов.

2.2 Decorator

Декоратор (Decorator) — структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет хорошую и гибкую альтернативу практике создания подклассов с целью расширения функциональности.



Используется для динамического подключения к объекту дополнительных обязательств.

Многие из вас спросят: как можно динамически (во время работы программы) добавить объекту новое поведение? Объект можно собрать из кусочков, то есть маленьких объектов. Помнишь цепочки фильтров в сервлетах? Или Stream API, когда ты писал запрос с использованием filter(), map(), list()?

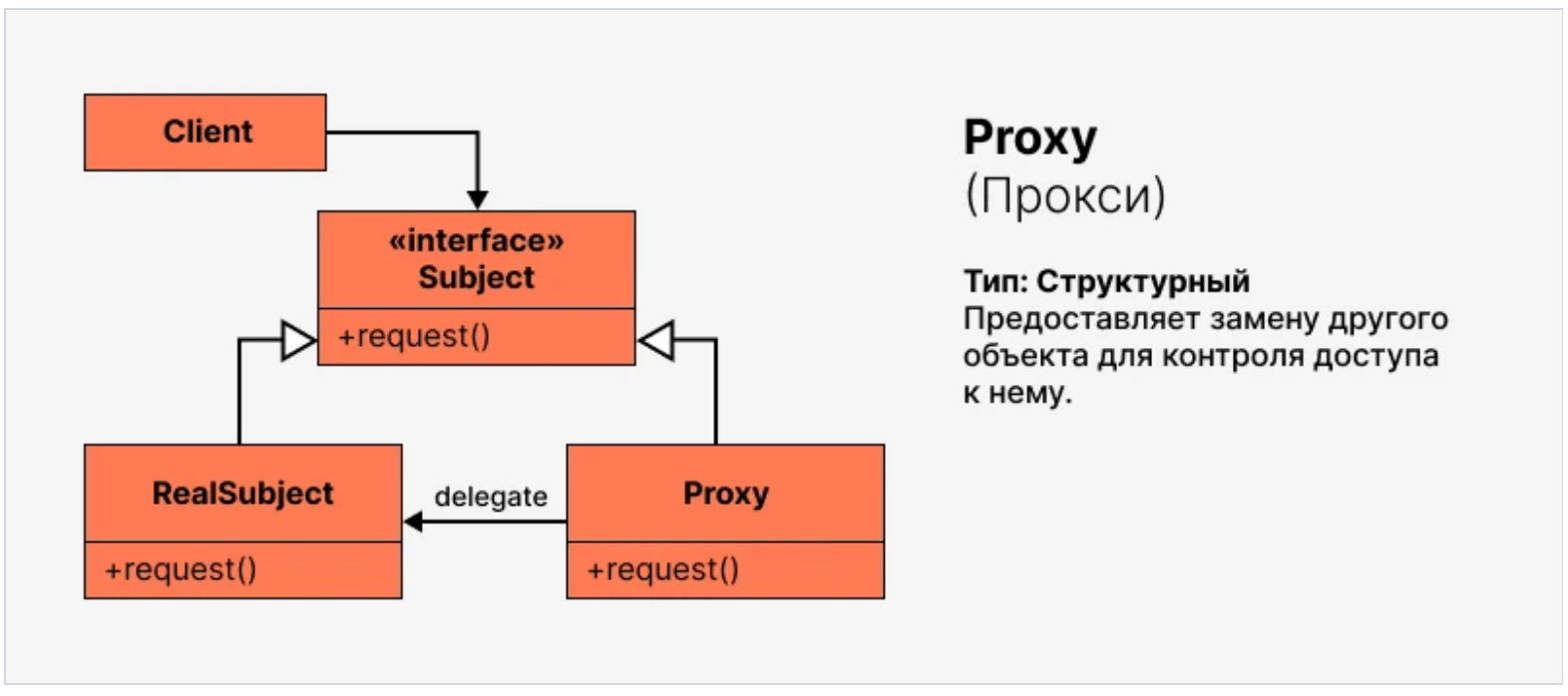
```
1  IntStream.of(50, 60, 70, 80, 90).filter(x -> x < 90).map(x -> x + 10).limit(3).forEach(System.out::print
```

Сильные стороны паттерна Decorator:

- Нет необходимости создавать подклассы для расширения функциональности объекта.
- Возможность динамически подключать новую функциональность в любом месте: до или после основной функциональности объекта ConcreteComponent.

2.3 Proxy

Заместитель (Proxy) — структурный шаблон проектирования, предоставляющий объект, который контролирует доступ к другому объекту, перехватывая и пропуская через себя все его вызовы.



Паттерн Proxy предоставляет объект-заменитель вместо настоящего объекта. Этот объект контролирует доступ к оригинальному объекту. Используется очень часто.

Помнишь, как мы использовали фреймворк Mockito и перехватывали обращение к реальному объекту с помощью метода Mockito.spy() или аннотации @Spy? Именно тогда и создавался специальный Proxy-объект, через который проходили все вызовы к оригинальному объекту.

И мы тогда с помощью добавления объекту правил могли этими вызовами управлять. Именно так – оригинальный объект не меняется, а работа с ним становится значительно гибче. Особенно полезно бывает, когда не мы из нашего кода вызываем proxy-объект, а передаем его куда-то. Контролируя таким образом общение двух независимых от нас объектов.

Виды прокси по назначению:

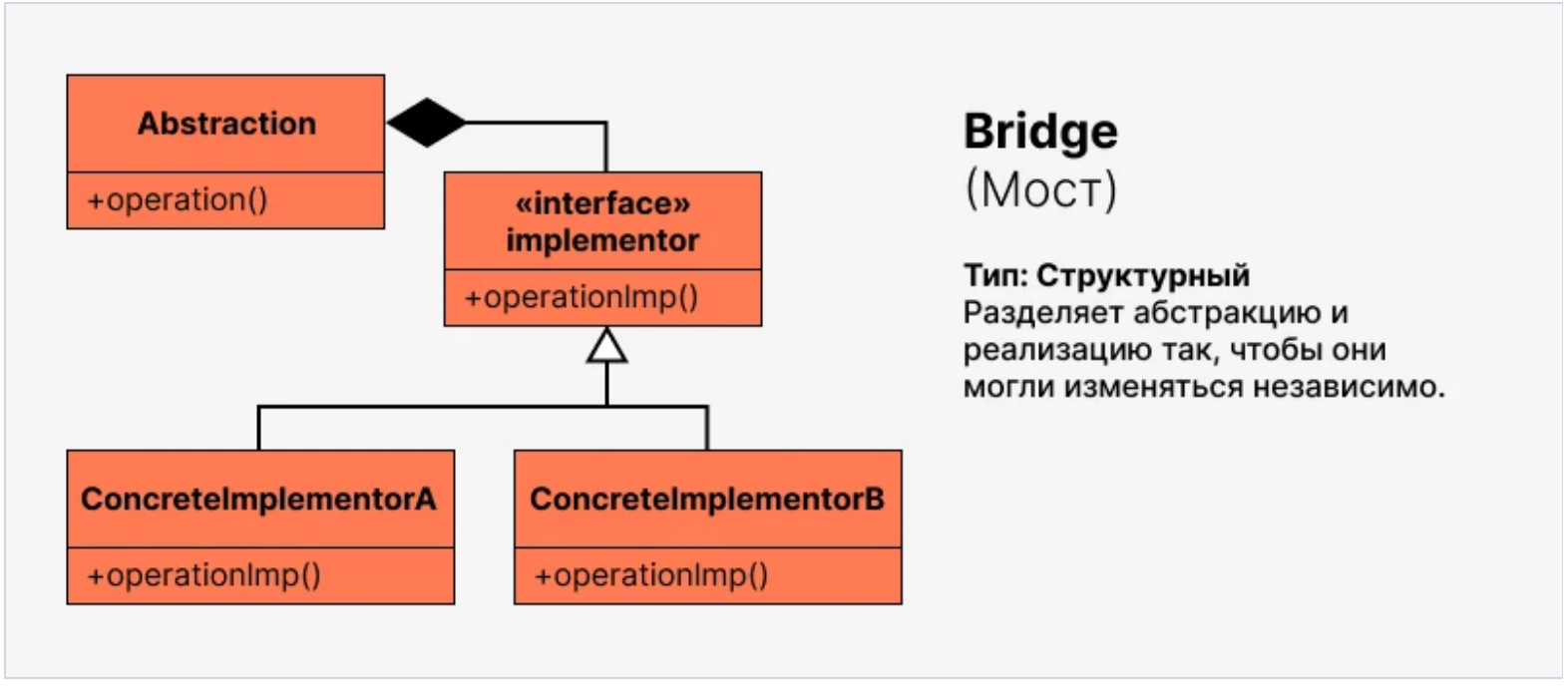
- **Протоколирующий прокси**: сохраняет в лог все вызовы “Субъекта” с их параметрами.

- **Удаленный заместитель** (remote proxies): обеспечивает связь с “Субъектом”, который находится в другом адресном пространстве или на удаленной машине. Также может отвечать за кодирование запроса и его аргументов и отправку закодированного запроса реальному “Субъекту”.
- **Виртуальный заместитель** (virtual proxies): обеспечивает создание реального “Субъекта” только тогда, когда он действительно понадобится. Также может кэшировать часть информации о реальном “Субъекте”, чтобы отложить его создание.
- **Копировать-при-записи**: обеспечивает копирование “субъекта” при выполнении клиентом определенных действий (частный случай “виртуального прокси”).
- **Защищающий заместитель** (protection proxies): может проверять, имеет ли вызывающий объект необходимые для выполнения запроса права.
- **Кэширующий прокси**: обеспечивает временное хранение результатов расчета до отдачи их множественным клиентам, которые могут разделить эти результаты.
- Экранирующий прокси: защищает “Субъект” от опасных клиентов (или наоборот).
- **Синхронизирующий прокси**: производит синхронизированный контроль доступа к “Субъекту” в асинхронной многопоточной среде.
- **“Умная” ссылка** (smart reference proxy): производит дополнительные действия, когда на “Субъект” создается ссылка, например, рассчитывает количество активных ссылок на “Субъект”.

2.4 Bridge

Шаблон Мост (Bridge) — структурный шаблон проектирования, используемый чтобы “разделять абстракцию и реализацию так, чтобы они могли изменяться независимо”.

Шаблон мост использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.



Когда абстракция и реализация разделены, они могут изменяться независимо. Другими словами, при реализации через шаблон мост, изменение структуры интерфейса не мешает изменению структуры реализации.

Рассмотрим такую абстракцию как фигура. Существует множество типов фигур, каждая со своими свойствами и методами. Однако есть что-то, что объединяет все фигуры. Например, каждая фигура должна уметь рисовать себя, масштабироваться и так далее.

В то же время рисование графики может отличаться в зависимости от типа ОС или графической библиотеки. Фигуры должны иметь возможность рисовать себя в различных графических средах. Но реализовывать в каждой фигуре все способы рисования или модифицировать фигуру каждый раз при изменении способа рисования непрактично.

В этом случае помогает шаблон мост, позволяя создавать новые классы, которые будут реализовывать рисование в различных графических средах. При использовании такого подхода очень легко можно добавлять как новые фигуры, так и способы их рисования.

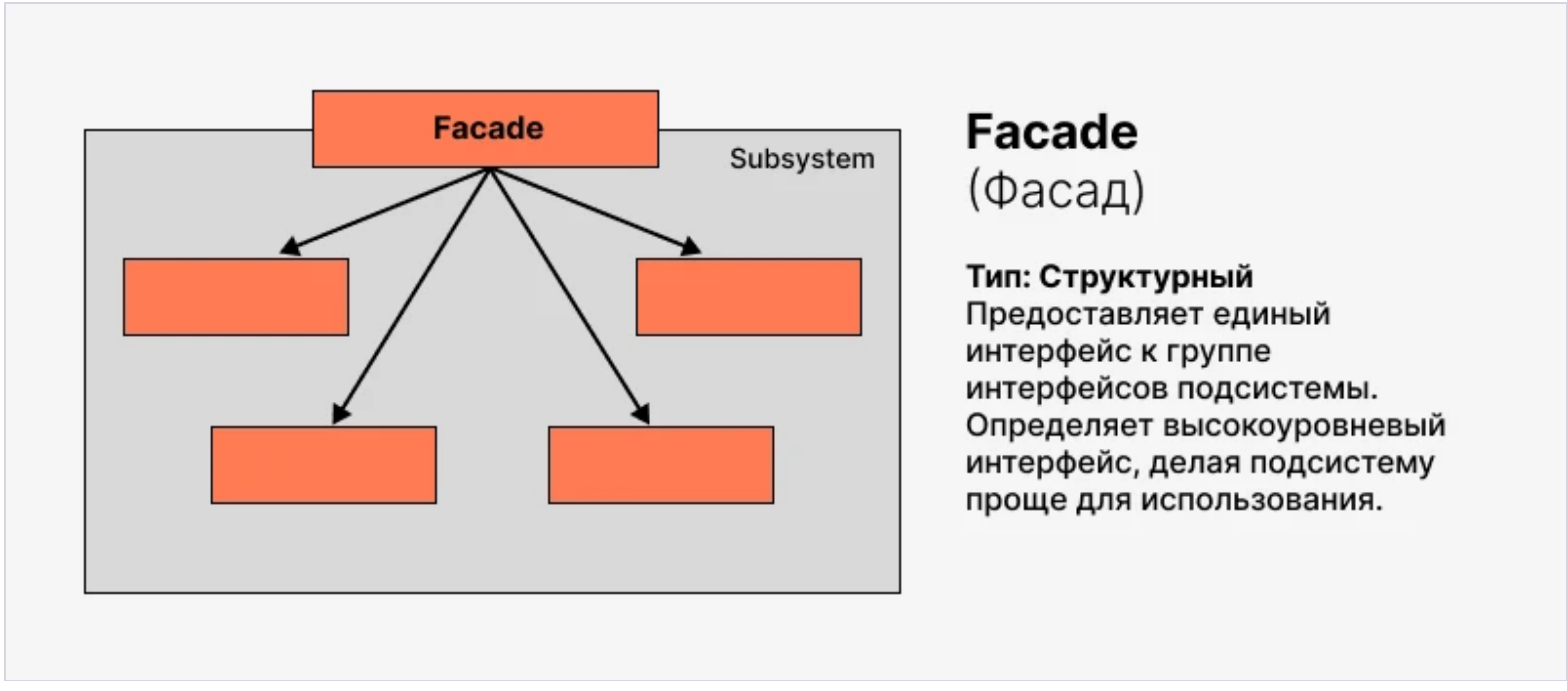
Связь, изображаемая стрелкой на диаграммах, может иметь 2 смысла: а) “разновидность”, в соответствии с принципом подстановки Лисков и б) одна из возможных реализаций абстракции. Обычно в языках используется наследование для реализации как а), так и б), что приводит к разбуханию иерархий классов.

Мост служит именно для решения этой проблемы: **объекты создаются парами** из объекта класса иерархии А и иерархии В, наследование внутри иерархии А имеет смысл “разновидность” по Лисков, а для понятия “реализация абстракции” используется

ссылка из объекта A в парный ему объект B.

2.5 Facade

Шаблон Фасад (Facade) — структурный шаблон проектирования, позволяющий скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.



Как обеспечить унифицированный интерфейс с набором разрозненных реализаций или интерфейсов, например, с подсистемой, если нежелательно сильное связывание с этой подсистемой или реализация подсистемы может измениться?

Определить одну точку взаимодействия с подсистемой — фасадный объект, обеспечивающий общий интерфейс с подсистемой, и возложить на него обязанность по взаимодействию с ее компонентами. Фасад — это внешний объект, обеспечивающий единственную точку входа для служб подсистемы.

Реализация других компонентов подсистемы закрыта и не видна внешним компонентам. Фасадный объект обеспечивает реализацию GRASP паттерна Устойчивый к изменениям с точки зрения защиты от изменений в реализации подсистемы.

Важно! Этот шаблон применяется, когда мы хотим полностью скрыть какую-то группу объектов и всю коммуникацию с ними пропустить через наш объект. Если же вы просто хотите обеспечить некоторый контроль процесса коммуникации объектов и скрывать их не обязательно, то лучше воспользоваться паттерном Proxy.

‹ Предыдущая лекция

Следующая лекция ›

Антон Уровень 90 EXPERT — 19 октября 2022, 23:54

С одной стороны, задачи классные. Сделаны с хорошим юмором, большой фантазией и глубоким пониманием автором задач материала.
С другой стороны, тема паттернов - трудная, теория на уровне дана сверх-кратко и обзорно (да, *"читай GoF book"*, но где бы еще и на это время взять...), и множество не относящихся к паттернам деталей и кода в задачах очень сбивает с толку. Не всегда можно сразу определить, что в коде просто для красоты задачи, а что - предметно и действительно нужно.
Подробный и проще написанный материал по паттернам + облегченные на детали задачи явно поспособствуют лучшему освоению этой непростой темы.
В подтверждение выше - судя по количеству решивших задачи (50 - 60 человек на момент написания этого комментария), многие студенты JRU даже не берутся за их решение.

Ответить 0

Gleb Уровень 48 — 9 октября 2022, 18:41

Очень непонятно написано. Особенно про мост. Я бы написал так:
Мост – это способ наделить разные объекты одного класса различным поведением путём хранения некоторой логики не в виде метода этого класса, а в виде его поля (переменной-члена), имеющего тип интерфейса. В различных объектах используется разная реализация этого интерфейса, за счёт чего и достигается различное поведение.

Ответить +3