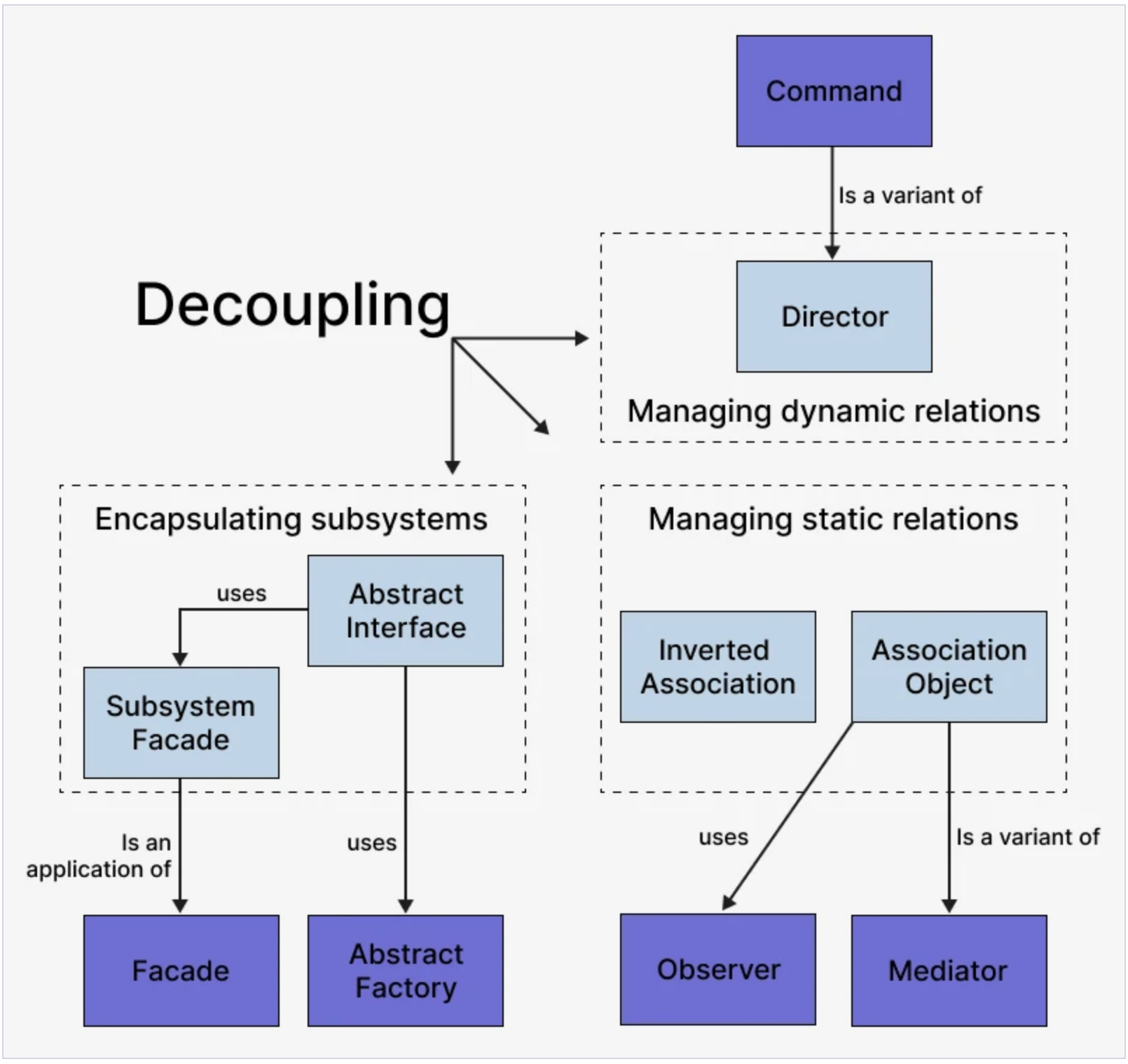
Как ослаблять связанность между модулями ПО

JSP & Servlets
14 уровень, 7 лекция

ОТКРЫТА

8.1 Декомпозиция – наше все

Для наглядности картинка из неплохой статьи "Decoupling of Object-Oriented Systems", иллюстрирующая основные моменты, о которых будет идти речь.



Тебе все еще кажется, что проектирование архитектуры приложения – это просто?

8.2 Интерфейсы, сокрытие реализации

Главными для уменьшения связанности системы являются принципы ООП и стоящий за ними принцип Инкапсуляция + Абстракция + Полиморфизм.

Именно поэтому:

- Модули должны быть друг для друга "черными ящиками" (инкапсуляция). Это означает, что один модуль не должен “лезть” внутрь другого модуля и что-либо знать о его внутренней структуре. Объекты одной подсистемы не должны обращаться напрямую к объектам другой подсистемы.
- Модули/подсистемы должны взаимодействовать друг с другом лишь посредством интерфейсов (то есть **абстракций**, не зависящих от деталей реализации). Соответственно каждый модуль должен иметь четко определенный интерфейс или интерфейсы для взаимодействия с другими модулями.

Принцип “черного ящика” (инкапсуляция) позволяет рассматривать структуру каждой подсистемы независимо от других подсистем. Модуль, представляющий собой “черный ящик”, можно относительно свободно менять. Проблемы могут возникнуть лишь на стыке разных модулей (или модуля и окружения).

И вот это взаимодействие нужно описывать в максимально общей (абстрактной) форме, то есть в форме интерфейса. В этом случае код будет работать одинаково с любой реализацией, соответствующей контракту интерфейса. Именно эта возможность работать с различными реализациями (модулями или объектами) через унифицированный интерфейс и называется полиморфизмом.

Именно поэтому Servlet – это интерфейс: веб-контейнер ничего не знает о сервлетах, для него это какие-то объекты, которые реализует интерфейс Servlet и все. Сервлеты тоже немного знают об устройстве контейнера. Интерфейс Servlet – это тот контракт, тот стандарт, то минимальное взаимодействие, которое нужно чтобы Java-веб-приложения завоевали мир.

Полиморфизм — это вовсе не переопределение методов, как иногда ошибочно полагают, а прежде всего — взаимозаменяемость модулей/объектов с одинаковым интерфейсом или «один интерфейс, множество реализаций». Для реализации полиморфизма механизм наследования совсем не нужен. Это важно понимать, поскольку наследования вообще, по возможности, следует избегать.

Благодаря интерфейсам и полиморфизму как раз и достигается возможность модифицировать и расширять код без изменения того, что уже написано (Open-Closed Principle).

До тех пор, пока взаимодействие модулей описано исключительно в виде интерфейсов и не завязано на конкретные реализации, ты имеешь возможность абсолютно “безболезненно” для системы заменить один модуль на любой другой, реализующий тот же интерфейс, а также добавить новый и тем самым расширить функциональность.

Это как в конструкторе LEGO — интерфейс стандартизирует взаимодействие и служит своего рода коннектором, куда может быть подключен любой модуль с подходящим разъемом.

Гибкость конструктора обеспечивается тем, что мы можем просто заменить одни модули или детали на другие с такими же разъемами (с тем же интерфейсом), а также добавить сколько угодно новых деталей (при этом уже существующие детали никак не изменяются и не переделываются).

Интерфейсы позволяют строить более простую систему, рассматривая каждую подсистему как единое целое и игнорируя ее внутреннее устройство. Они дают возможность модулям взаимодействовать и при этом ничего не знать о внутренней структуре друг друга, тем самым в полной мере реализуя принцип минимального знания, являющегося основой слабой связанности.

Чем в более общей/абстрактной форме определены интерфейсы и чем меньше ограничений они накладывают на взаимодействие, тем гибче система. Отсюда фактически следует еще один из принципов SOLID — **Принцип разделения интерфейса** (Interface Segregation Principle), который выступает против “толстых интерфейсов”.

Он говорит, что большие, объемные интерфейсы надо разбивать на более маленькие и специфические, чтобы клиенты маленьких интерфейсов (зависящие модули) знали только о методах, которые необходимы им в работе.

Формулируется этот принцип следующим образом: "Клиенты не должны зависеть от методов (знать о методах), которые они не используют" или “Много специализированных интерфейсов лучше, чем один универсальный”.

Получается, что слабая связность обеспечивается лишь тогда, когда взаимодействие и зависимости модулей описываются лишь с помощью интерфейсов, то есть абстракций, без использования знаний об их внутреннем устройстве и структуре.И фактически тем самым реализуется инкапсуляция. Плюс мы имеем возможность расширять/изменять поведения системы за счет добавления и использования различных реализаций, то есть за счет полиморфизма. Да, мы опять пришли к ООП — Инкапсуляция, Абстракция, Полиморфизм.

8.3 Фасад: интерфейс модуля

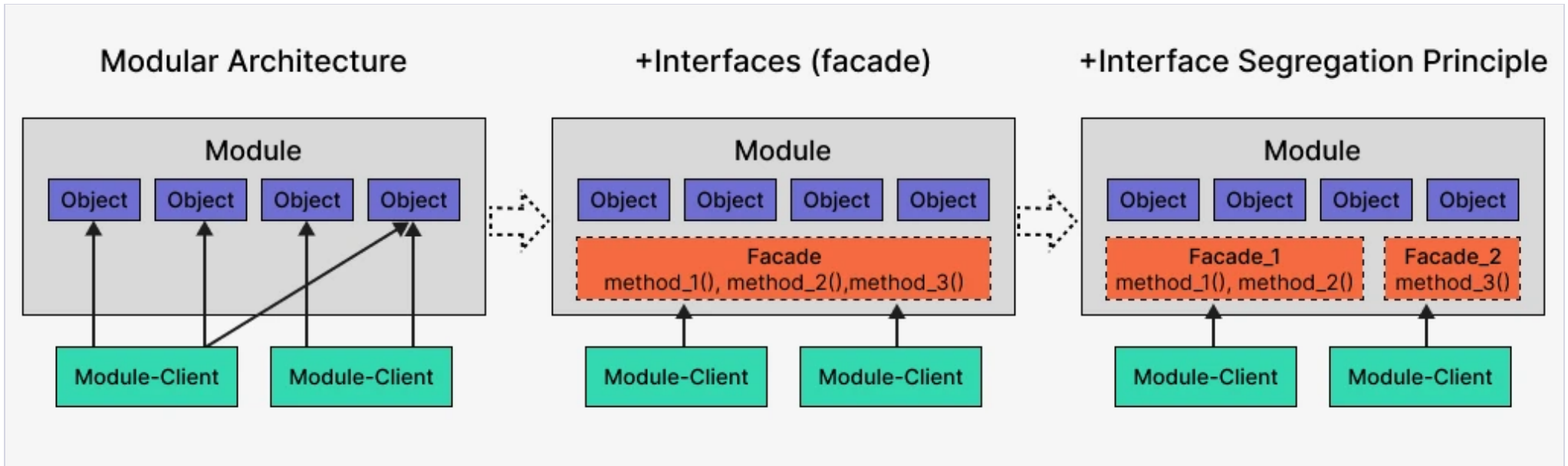
Тут опытный программист спросит: если проектирование идет не на уровне объектов, которые сами же и реализуют соответствующие интерфейсы, а на уровне модулей, то что является реализацией интерфейса модуля?

Ответ: если говорить языком паттернов проектирования, то за реализацию интерфейса модуля может отвечать специальный объект — **Фасад**. Если ты вызываешь методы объекта, который содержит суффикс Gateway (например, MobileApiGateway), то, скорее всего, это фасад.

Фасад — это объект-интерфейс, аккумулирующий в себе высокоуровневый набор операций для работы с некоторой подсистемой, скрывающий за собой ее внутреннюю структуру и истинную сложность. Обеспечивает защиту от изменений в реализации подсистемы. Служит единой точкой входа — "ты пинаешь фасад, а он знает, кого там надо пнуть в этой подсистеме, чтобы получить нужное".

Ты только что познакомился с одним из самых важных паттернов проектирования, позволяющим использовать концепцию интерфейсов при проектировании модулей и тем самым ослаблять их связанность — “Фасад”.

Помимо этого, “Фасад” дает возможность работать с модулями точно также как с обычными объектами и применять при проектировании модулей все те полезные принципы и техники, которые используются при проектировании классов.



Замечание: хотя большинство программистов понимают важность интерфейсов при проектировании классов (объектов), складывается впечатление, что идею необходимости использовать интерфейсы также и на уровне модулей многие открывают сами.

‹ Предыдущая лекция

Следующая лекция ›