Лекция 8

Шаблоны проектирования - UML в действии.

План

- Шаблоны разработки ПО
- Стиль мышления эксперта
- Определение шаблона
- Свойства шаблонов
- Типы паттернов
- Типы шаблонов разработки ПО
- Паттерны проектирования
- Классификация по целям паттернов проектирования

Шаблоны разработки ПО

- В разработке ПО часто встречаются проблемы, которые уже решались ранее в других проектах.
- В связи с тем, что контексты, в которых данная проблема решалась, могут различаться
  - (другой тип приложения, другая платформа или другой язык программирования),
  - все обычно заканчивается повторением проектирования и реализации данного решения,
  - тем самым возникает ситуация «повторного изобретения колеса».

Стиль мышления эксперта

- При решении конкретных проблем ***эксперты*** обычно не пытаются разработать новое решение, который отличается от уже имеющихся.
- Действия эксперта:
  - вспоминают аналогичную проблему, которую они уже решали,
  - стараются повторно используют суть ранее принятого решения для решения новой проблемы.
- Такой «стиль мышление» в терминах пар «проблема - решение», является общим для множества различных предметных областей, таких, как:
  - архитектура;
  - экономика;
  - программная инженерия.

Зачем нужны шаблоны

- Шаблоны позволяют основываться на коллективном опыте квалифицированных инженеров по проектированию.
- Они фиксируют существующий, хорошо-зарекомендовавший себя опыт разработки и помогает содействовать хорошим методам проектирования.
- Каждый шаблон имеет дело с конкретной, многократно встречающейся проблемой в области проектирования и реализации.

Определение шаблона

- Шаблон это описание хорошо проверенной, обобщенной схемы решения некоторой часто повторяющейся проблемы (задачи) разработки ПО, которая возникает в некоторых специфических условиях (контексте).
- Схема решения проблемы задается путем
  - определения используемых (составляющих) компонент;
  - их ответственностей;
  - способов их взаимодействия.

Свойства шаблонов

1. ***Шаблоны описывают решения*** для часто повторяющихся задач проектирования, которая возникают в некоторых специфических ситуациях.
2. ***Шаблоны документируют*** накопленный, хорошо зарекомендовавший себя опыт проектирования.
3. ***Шаблоны определяют и описывают абстракции***, которые находятся на более высоком уровне, чем уровень отдельных классов и экземпляров или компонентов.
4. ***Шаблоны предоставляют общий словарь терминов*** и общее понимание принципов проектирования.

Свойства шаблонов (2)

5. ***Шаблоны являются средствами документирования архитектур*** ПО.
6. ***Шаблоны поддерживают конструирование ПО с определенными свойствами.***
7. ***Шаблоны помогают разрабатывать сложные и разнородные архитектуры*** ПО.
8. ***Шаблоны помогают справиться с сложностью*** ПО.

Типы паттернов

- В ОО анализе и проектировании разработано много различных паттернов.

1. Архитектурные паттерны.
   - Описывают фундаментальные способы структурирования программных систем.
   - Эти паттерны относятся к уровню систем и подсистем, а не классов.
2. Паттерны проектирования.
   - Описывают структуру программных систем в терминах классов.
   - Наиболее известными в этой области являются 23 паттерна, описанные в [GoF].
3. Паттерны анализа.
   - Представляют общие схемы организации процесса объектно-ориентированного моделирования.

Типы шаблонов разработки ПО

- На этапе анализа системы:
  - ***шаблоны анализа*** (analysis patterns) – комбинации классов для описания стандартных задач прикладной области;
- На этапе проектирования системы
  - ***шаблоны архитектуры*** (architecture patterns).
  - ***шаблоны проектирования*** (design patters);
  - специфичные для конкретного языка программирования ***идиомы***.

Паттерны проектирования (Design patterns)

- Специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.
- Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы, которые реализуют исходную проблему проектирования в конкретном контексте. Наиболее известными паттернами этой категории являются паттерны GoF (Gang of Four), названные в честь Э. Гаммы, Р. Хелма, Р. Джонсона и Дж. Влиссидеса, которые систематизировали их и представили общее описание. Паттерны GoF включают в себя 23 паттерна. Эти паттерны не зависят от языка реализации, но их реализация зависит от области приложения.

Понятие паттерна проектирования

- **Паттерн проектирования ПО – это описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте [GoF].**
- Следует отличать паттерны проектирования от идиом.
  - Паттерны проектирования не зависят от выбора языка (хотя их реализации, зачастую, зависимы от языка программирования).
  - Идиомы — это паттерны, описывающие типичные решения на конкретном языке программирования.

Причины возникновения паттернов проектирования

- В конце 80-х годов XX века в области разработки ПО (в частности, ОО проектировании) накопилось много различных похожих по своей сути решений.
- Эти решения требовали
  - систематизации,
  - обобщения на всевозможные ситуации,
  - доступного описания, способствующего пониманию их людьми, которые до этого никогда их не использовали.
- Такое упорядочение знаний в ОО проектировании позволило бы повторно использовать готовые и уже проверенные решения, а не снова и снова «изобретать велосипед».
- Решение данной проблемы взяли на себя паттерны проектирования.

Описание паттернов

- В общем случае каждый паттерн состоит из таких составляющих:

1. **Имя** является уникальным идентификатором паттерна. Имена паттернов проектирования, описанных в [GoF], являются общепринятыми.
2. **Задача** описывает ситуацию, в которой можно применять паттерн.
3. **Решения задачи** проектирования в виде паттерна определяет общие функции каждого элемента дизайна и отношения между ними.
4. **Результаты** представляют следствия применения паттерна.

Практическое применение паттернов проектирования

- Паттерны проектирования представляют общие решения типичных задач объектно-ориентированного проектирования.
- Не обязательно для хорошего дизайна системы проектировать её части на основе паттернов проектирования,
  - Специально подводить дизайн ПО под уже известные шаблоны.
- С практической точки зрения, отталкиваться от паттернов проектирования не является самым эффективным и гибким подходом.
  - Это может привести к решению не отвечающему требованиям гибкости и масштабируемости решению.

Базовые принципы гибкого проектирования

- Всегда формировать простой дизайн: из двух предложенных решений, как правило, лучшим является то, что проще.
- Слабая зависимость: *дизайн модуля должен быть таким, чтобы в случае его модификации зависимые фрагменты системы не требовали (или почти не требовали) изменений.*
- При таком подходе к проектированию логической структуры системы *можно увидеть типичные задачи, которые уже решены с помощью паттернов проектирования.*

- В этом случае целесообразно применить уже готовое шаблонное решение (паттерн),
  - при этом нужно оценить возможности и перспективы на основании его описания.
- Не следует забывать о принципах простого дизайна и слабой зависимости,
  - излишнее желание воспользоваться паттерном проектирования может способствовать формированию плохого дизайна.
- Дизайн программной системы постоянно развивается -- успешно примененный паттерн со временем может трансформироваться в совсем другой или вовсе исчезнуть.

Унификация терминологии

- Благодаря паттернам проектирования произошла:
  - ~~ссылки на паттерны проектирования, как на известные решения~~ задач, можно добавлять в проектную документацию, а также использовать в дискуссиях.
- Практический опыт разработки программного обеспечения говорит, что любой полученный результат всегда подлежит тщательной проверке.

Организация каталога шаблонов проектирования ПО

- Паттерны проектирования различаются степенью детализации и уровнем абстракции и должны быть каким-то образом организованы.
- Создана классификация, позволяющая ссылаться на семейства взаимосвязанных паттернов.
- Она позволяет
  - быстрее освоить паттерны, описываемые в каталоге,
  - указывает направление поиска новых.

Критерии классификации

- Паттерны можно классифицировать по двум критериям :
  - Цель -- отражает назначение паттерна.
  - Уровень -- говорит о том, к чему обычно применяется паттерн.

Классификация по целям паттернов проектирования

1. **Порождающие паттерны** (Creational Patterns).
   - Определяют способы создания объектов в системе.
2. **Структурные паттерны** (Structural Patterns).
   - Описывают способы построение сложных структур из классов и объектов.
3. **Поведенческие паттерны** (Behavioral Patterns).
   - Описывают способы взаимодействия между объектами.

Список паттернов проектирования (порождающие)

- Factory Method (фабричный метод)
  - Определяет интерфейс для создания объектов, при этом выбранный класс инстанцируется подклассами.
- Abstract Factory (абстрактная фабрика)
  - Предоставляет интерфейс для создания семейств, связанных между собой, или независимых объектов, конкретные классы которых неизвестны.
- Singleton (одиночка)
  - Гарантирует, что некоторый класс может иметь только один экземпляр, и предоставляет глобальную точку доступа к нему.
- Prototype (прототип)
  - Описывает виды создаваемых объектов с помощью прототипа и создает новые объекты путем его копирования.
- Builder (строитель)
  - Отделяет конструирование сложного объекта от его представления, позволяя использовать один и тот же процесс конструирования для создания различных представлений.

Список паттернов проектирования (структурные)

- Adapter (адаптер)
  - Преобразует интерфейс класса в некоторый другой интерфейс, ожидаемый клиентами.
  - Обеспечивает совместную работу классов, которая была бы невозможна без данного паттерна из-за несовместимости интерфейсов.
- Bridge (мост)
  - Отделяет абстракцию от реализации, благодаря чему появляется возможность независимо изменять то и другое.
- Decorator (декоратор)
  - Динамически возлагает на объект новые функции.
  - Применяются для расширения имеющейся функциональности и являются гибкой альтернативой порождению подклассов.

Список паттернов проектирования (структурные) (2)

- Proxy (заместитель)
  - Подменяет другой объект для контроля доступа к нему
- Facade (фасад)
  - Предоставляет унифицированный интерфейс к множеству интерфейсов в некоторой подсистеме.
  - Определяет интерфейс более высокого уровня, облегчающий работу с подсистемой.
- Composite (компоновщик)
  - Группирует объекты в древовидные структуры для представления иерархий типа «часть-целое».
  - Позволяет клиентам работать с единичными объектами так же, как с группами объектов.
- Flyweight (приспособленец)
  - Использует разделение для эффективной поддержки большого числа мелких объектов.

Список паттернов проектирования (поведения)

- Interpreter (интерпретатор)
  - Для заданного языка определяет представление его грамматики, а также интерпретатор предложений языка, использующий это представление.
- Template Method (шаблонный метод)
  - Определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы.
  - Позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.
- Iterator (итератор)
  - Дает возможность последовательно обойти все элементы составного объекта, не раскрывая его внутреннего представления.

Список паттернов проектирования (поведения) (2)

- Command (команда)
  - Инкапсулирует запрос в виде объекта, позволяя тем самым параметризовывать клиентов типом запроса, устанавливать очередность запросов, протоколировать их и поддерживать отмену выполнения операций.
- Observer (наблюдатель)
  - Определяет между объектами зависимость типа один-ко-многим, так что при изменении состоянии одного объекта все зависящие от него получают извещение и автоматически обновляются.
- Visitor (посетитель)
  - Представляет операцию, которую надо выполнить над элементами объекта. Позволяет определить новую операцию, не меняя классы элементов, к которым он применяется.

Список паттернов проектирования (поведения) (3)

- Mediator (посредник)
  - Определяет объект, в котором инкапсулировано знание о том, как взаимодействуют объекты из некоторого множества.
  - Способствует уменьшению числа связей между объектами, позволяя им работать без явных ссылок друг на друга.
  - Это, в свою очередь, дает возможность независимо изменять схему взаимодействия.
- Memento (хранитель)
  - Позволяет, не нарушая инкапсуляции, получить и сохранить во внешней памяти внутреннее состояние объекта, чтобы позже объект можно было восстановить точно в таком же состоянии.

Список паттернов проектирования (5)

- State (состояние)
  - Позволяет объекту варьировать свое поведение при изменении внутреннего состояния.
  - При этом создается впечатление, что поменялся класс объекта.
  - Strategy (стратегия)
  - Определяет семейство алгоритмов, инкапсулируя их все и позволяя подставлять один вместо другого.
  - Можно менять алгоритм независимо от клиента, который им пользуется.

Список паттернов проектирования (поведения) (4)

- Chain of Responsibility (цепочка обязанностей)
  - Позволяет избежать жесткой зависимости отправителя запроса от его получателя, при этом запросом начинает обрабатываться один из нескольких объектов.
  - Объекты-получатели связываются в цепочку, и запрос передается по цепочке, пока какой-то объект его не обработает.

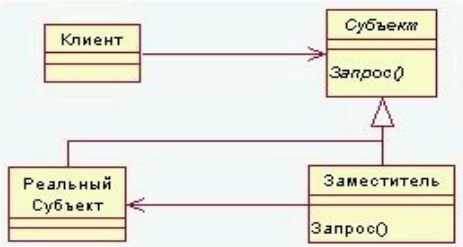
Абстрактная фабрика

Проблема	Создать семейство взаимосвязанных или взаимозависимых обьектов (не специфицируя их конкретных классов).
Решение	Создать абстрактный класс, в котором объявлен интерфейс для создания конкретных классов.
Пример	Какой класс должен отвечать за создание обьектов - адаптеров при использовании паттерна "Адаптер", см. 3.1.1. Если подобные объекты создаются неким объектом уровня предметной области, то будет нарушен принцип разделения обязанностей.
Преимущества	Изолирует конкретные классы. Поскольку "Абстрактная фабрика" инкапсулирует ответственность за создание классов и сам процесс их создания, то она изолирует клиента от деталей реализации классов. Упрощена замена "Абстрактной фабрики", поскольку она используется в приложении только один раз при инстанцировании.
Недостатки	Интерфейс "Абстрактной фабрики" фиксирует набор обьектов, которые можно создать. Расширение "Абстрактной фабрики" для изготовления новых обьектов часто затруднительно.

Одиночка (Singleton)

Проблема	Какой специальный класс должен создавать "Абстрактную фабрику", см. 3.3.1 и как получить к ней доступ? Необходим лишь один экземпляр специального класса, различные объекты должны обращаться к этому экземпляру через единственную точку доступа.
Решение	Создать класс и определить статический метод класса, возвращающий этот единственный объект.
Рекомендации	Разумнее создавать именно статический экземпляр специального класса, а не объявить требуемые методы статическими, поскольку при использовании методов экземпляра можно применить механизм наследования и создавать подклассы. Статические методы в языках программирования не полиморфны и не допускают перекрытия в производных классах. Решение на основе создания экземпляра является более гибким, поскольку впоследствии может потребоваться уже не единственный экземпляр объекта, а несколько.

Заместитель (Proxy) или Суррогат (Surrogate)

Проблема	Необходимо управлять доступом к объекту, так чтобы создавать громоздкие объекты "по требованию".
Решение	Создать суррогат громоздкого объекта. "Заместитель" хранит ссылку, которая позволяет заместителю обратиться к реальному субъекту (объект класса "Заместитель" может обращаться к объекту класса "Субъект", если интерфейсы "РеальногоСубъекта" и "Субъекта" одинаковы). Поскольку интерфейс "РеальногоСубъекта" идентичен интерфейсу "Субъекта", так, что "Заместителя" можно подставить вместо "РеальногоСубъекта", контролирует доступ к "РеальномуСубъекту", может отвечать за создание или удаление "РеальногоСубъекта". "Субъект" определяет общий для "РеальногоСубъекта" и "Заместителя" интерфейс, так, что "Заместитель" может быть использован везде, где ожидается "РеальныйСубъект". При необходимости запросы могут быть переадресованы "Заместителем" "РеальномуСубъекту".  "Заместитель" может иметь и другие обязанности, а именно: • удаленный "Заместитель" может отвечать за кодирование запроса и его аргументов и отправку закодированного запроса реальному "Субъекту", • виртуальный "Заместитель" может кэшировать дополнительную информацию о реальном "Субъекте", чтобы отложить его создание, • защищающий "Заместитель" может проверять, имеет ли вызывающий объект необходимые для выполнения запроса права.

Адаптер

Проблема	Необходимо обеспечить взаимодействие несовместимых интерфейсов или как создать единый устойчивый интерфейс для нескольких компонентов с разными интерфейсами.
Решение	Конвертировать исходный интерфейс компонента к другому виду с помощью промежуточного объекта - адаптера, то есть, добавить специальный объект с общим интерфейсом в рамках данного приложения и перенаправить связи от внешних обьектов к этому объекту - адаптеру.
Пример	Соответствует примеру из описания паттерна "Полиморфизм", см. п. 3.2.15.

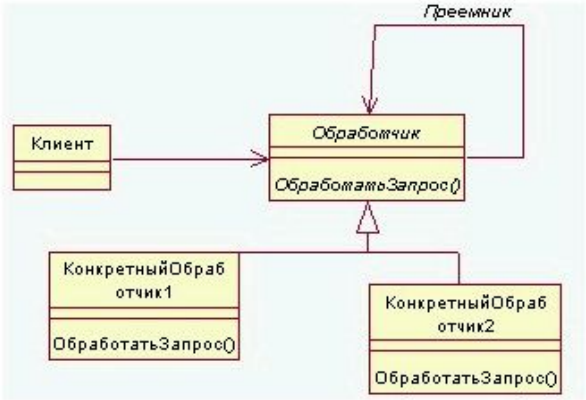
Composite – Компоновщик

Проблема	Как обрабатывать группу или композицию структур обьектов одновременно?
Решение	Определить классы для композитных и атомарных обьектов таким образом, чтобы они реализовывали один и тот же интерфейс.
Пример	См. паттерн "Стратегия", 3.2.9, необходимо учесть несколько скидок различных видов (зависят от времени, типа покупателя, типом выбранного продукта. Как применять политику ценообразования? Вырабатывается стратегия приоритета скидок, объект "Продажа" не должен обладать информацией о применяемых скидках, но можно было бы применить стратегию расчета скидок. Создается новый класс "РасчетСкидкиАлгоритмКомпозит".

Facade - Фасад

Проблема	Как обеспечить унифицированный интерфейс с набором разрозненных реализаций или интерфейсов, например, с подсистемой, если нежелательно высокое связывание с этой подсистемой или реализация подсистемы может измениться?
Решение	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 паттерна "Устойчивый к изменениям" с точки зрения защиты от изменений в реализации подсистемы., см. п. 3.1.9.

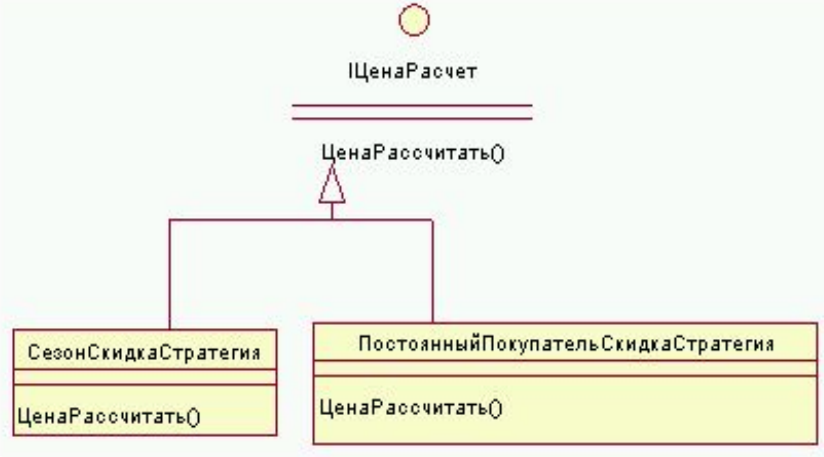
Chain of Responsibility - Цепочка обязанностей

Проблема	Запрос должен быть обработан несколькими объектами.
Рекомендации	Логично использовать данный паттерн, если имеется более одного объекта, способного обработать запрос и обработчик заранее неизвестен (и должен быть найден автоматически) или если весь набор обьектов, которые способны обработать запрос, должен задаваться автоматически.
Решение	Связать объекты - получатели запроса в цепочку и передать запрос вдоль этой цепочки, пока он не будет обработан. "Обработчик" определяет интерфейс для обработки запросов, и, возможно, реализует связь с преемником, "КонкретныйОбработчик" обрабатывает запрос, за который отвечает, имеет доступ к своему преемнику ("КонкретныйОбработчик" направляет запрос к своему преемнику, если не может обработать запрос сам. 
Преимущества	Ослабляется связанность (объект не обязан "знать", кто именно обработает его запрос).
Недостатки	Нет гарантий, что запрос будет обработан, поскольку он не имеет явного получателя.

Observer

Проблема	Один объект ("Подписчик") должен знать об изменении состояний или некоторых событиях другого объекта. При этом необходимо поддерживать низкий уровень связывания с объектом - "Подписчиком".
Решение	Определить интерфейс "Подписки". Объекты - подписчики реализуют этот интерфейс и динамически регистрируются для получении информации о некотором событии. Затем при реализации условленного события оповещаются все объекты - подписчики.

Strategy – Стратегия

Проблема	Спроектировать изменяемые, но надежные алгоритмы или стратегии.
Решение	Определить для каждого алгоритма или стратегии отдельный класс со стандартным интерфейсом.
Пример	Обеспечение сложной логики вычисления стоимости товаров с учетом сезонных скидок, скидок постоянным клиентам и т. п. Данная стратегия может изменяться.  Создается несколько классов "Стратегия", каждый из которых содержит один и тот же полиморфный метод "ЦенаРассчитать". В качестве параметров в этот метод передаются данные о продаже. Объект стратегии связывается с контекстным объектом (тем объектом, к которому применяется алгоритм).

Template Method - Шаблонный метод

Проблема	Определить алгоритм и реализовать возможность переопределения некоторых шагов алгоритма для подклассов (без изменения общей структуры алгоритма.
Решение	"АбстрактныйКласс" определяет абстрактные Операции(), замещаемые в конкретных подклассах для реализации шагов алгоритма, и реализует ШаблонныйМетод(), определяющий "скелет" алгоритма. "КонкретныйКласс" релизует Операции(), выполняющие шаги алгоритма способом, который зависит от подкласса. "КонкретныйКласс" предполагает, что инвариантные шаги алгоритма будут выполнены в "АбстрактномКлассе".

Контрольные вопросы

- Дайте определение шаблона разработки ПО
- Назовите свойства шаблонов
- Какие типы паттернов вы знаете?
- Типы шаблонов разработки ПО
- Что такое паттерны проектирования?
- Перечислите классификацию паттернов проектирования по целям
- Назовите порождающие паттерны проектирования
- Назовите структурные паттерны проектирования
- Назовите поведенческие паттерны проектирования

Литература

- Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма и др. - Москва: СИНТЕГ, 2016. - 366 с.
- Дж. Рамбо, М. Блаха. UML 2.0 Объектно-ориентированное моделирование и разработка. 2-е издание. СПб.: Питер, 2007 - 544 с.; ил.