Паттерны проектирования

JSP & Servlets
16 уровень, 0 лекция

ОТКРЫТА

1.1 Знакомство с паттернами

Как уже упоминалось ранее, программист начинает работу над программой с проектирования ее модели: составления списка сущностей, которыми будет оперировать программа. И чем больше в программе сущностей, тем сложнее программа.

Поэтому чтобы снизить сложность программы, взаимодействия объектов стараются стандартизировать. И в этом программисту очень сильно помогают **шаблоны проектирования** или **паттерны** проектирования. От английского design **pattern**.

Важно! В русском языке под словом дизайн обычно подразумевают графический дизайн, в английском это не так. Английское слово design по смыслу ближе к к слову “проектирование” и/или “устройство”. Например, дизайн двигателя — это не его внешний вид, а его внутреннее устройство.

Поэтому design pattern – это именно паттерн/шаблон проектирования. Рекомендую вообще перестать употреблять слово дизайн в смысле “внешний вид”. Ты – будущий Software Engineer, и для тебя дизайн – это именно проектирование.

Так что же такое этот design pattern? В первую очередь паттерн проектирования – это **стандартное решение стандартной проблемы**. Хорошее, эффективное и проверенное временем решение.

Допустим, тебе задали спроектировать велосипед, ты можешь сделать ему два колеса, три или даже пять. Так кстати на заре проектирования и было. Но проверенный временем подход – два колеса. А ведь к нынешнему очевидному подходу шли через боль и ошибки:

Обычно шаблон не является законченным решением, который может быть прямо преобразован в код, это лишь пример хорошего решения задачи, который можно использовать в различных ситуациях.

Объектно-ориентированные шаблоны **показывают отношения и взаимодействия между классами или объектами**, без определения того, какие конечные классы или объекты приложения будут использоваться.

1.2 История появления паттернов проектирования

Еще в 70-е годы программисты столкнулись с необходимостью разрабатывать большие программы, над которыми должны были трудиться целые команды разработчиков. Были испробованы различные методы организации работы, но сильнее всего на разработку повлияла строительная сфера.

Для организации работы большой группы людей использовали практики и подходы из сферы строительства. Кстати, именно оттуда в программирование пришли такие термины как сборка (build), Software Developer (строитель), и понятие архитектуры.

И как ты уже догадываешься, идею design pattern тоже почерпнули из сферы строительства. Концепцию паттернов впервые описал Кристофер Александер в книге “Язык шаблонов. Города. Здания. Строительство”. В этой книге для описания процессов проектирования городов был использован специальный язык – паттерны.

Паттерны в строительстве описывали типичные проверенные временем решения: какой высоты сделать окна, сколько этажей должно быть в здании, сколько площади в микрорайоне отвести под деревья и газоны.

Поэтому ничего удивительного, что в 1994 году вышла книга “Приемы объектно-ориентированного проектирования. Паттерны проектирования”, в которую вошли 23 паттерна, решающие различные проблемы объектно-ориентированного дизайна.

Книгу написали 4 автора: Эрих Гамма, Ричард Хелм, Ральф Джонсон и Джон Влиссидес. Название книги было слишком длинным, чтобы кто-то смог его запомнить. Поэтому вскоре все стали называть её “book by the gang of four”, то есть **“книга от банды четырех”**, а затем и вовсе “GoF book”.

И с тех пор были открыты еще другие паттерны проектирования. “Паттерновый” подход стал популярен во всех областях программирования, поэтому сейчас можно встретить всевозможные паттерны и за пределами объектного проектирования.

Важно! Паттерны — это не какие-то супер-оригинальные решения, а наоборот, часто встречающиеся, типовые решения одной и той же проблемы. Хорошие проверенные временем решения.

1.3 Список паттернов

Многие программисты за всю жизнь не изучили ни одного паттерна, что, впрочем, не мешает им их применять. Как мы уже говорили раньше, паттерны – это хорошие проверенные временем решения и, если программист не дурак, то с опытом сам находит такие решения.

Но зачем через десятки проб и ошибок приходить к оптимальным решениям, когда есть люди, которые уже прошли этот путь и написали книги с квинтэссенцией полученного опыта и жизненной мудрости?

Можно забить гвоздь гаечным ключом, но зачем? Можно даже и дрелью, если сильно постараться. Но хорошее осознанное владение инструментом как раз и отличает профессионала от любителя. И профессионал знает, что главная фишка дрели совсем не в этом. Итак, зачем же знать паттерны?

- Проверенные решения. Ты тратишь меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений ты смог бы додуматься и сам, но многие могут быть для тебя открытием.
- Стандартизация кода. Ты делаешь меньше просчетов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.
- Общий программистский словарь. Ты произносишь название паттерна вместо того, чтобы час объяснять другим программистам, какой крутой дизайн ты придумал и какие классы для этого нужны.

Какие бывают паттерны?

Паттерны отличаются по уровню сложности, детализации и охвата проектируемой системы. Проводя аналогию со строительством, ты можешь повысить безопасность перекрестка, поставив светофор, а можешь заменить перекресток целой автомобильной развязкой с подземными переходами.

Самые низкоуровневые и простые паттерны — идиомы. Они не универсальны, поскольку применимы только в рамках одного языка программирования.

Самые универсальные — архитектурные паттерны, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных ее элементов.

Но главное – паттерны отличаются назначением. Паттерны, с которыми мы познакомимся, можно разбить на три основные группы:

- **Порождающие паттерны** заботятся о гибком создании объектов без внесения в программу лишних зависимостей.
- **Структурные паттерны** показывают различные способы построения связей между объектами.
- **Поведенческие паттерны** заботятся об эффективной коммуникации между объектами.

1.4 Знакомство с UML

Начнем с рассмотрения тех самых 23 паттернов, которые были описаны в книге банды четырех. И сами паттерны, и их названия – это знакомые вещи даже для начинающего программиста. Я тебя с ними познакомлю, но настоятельно рекомендую прочитать ту самую книгу про паттерны.

Паттерны проектирования не привязаны к конкретному языку программирования, так что для их описания обычно используется язык UML. Он был очень популярен лет 20 назад, но и сейчас им иногда пользуются. И кстати, описание паттернов – это как раз то место, где использование UML – стандарт.

С помощью UML ты можешь описать отношения между различными сущностями. В нашем случае – это объекты и классы.

Отношения между классами описываются четырьмя типами стрелочек:

	**композиция (composition)** — подвид агрегации, в которой “части” не могут существовать отдельно от “целого”.
	**агрегация (aggregation)** — описывает связь “часть”–”целое”, в котором “часть” может существовать отдельно от “целого”. Ромб указывается со стороны “целого”.
	**зависимость (dependency)** — изменение в одной сущности (независимой) может влиять на состояние или поведение другой сущности (зависимой). Со стороны стрелки указывается независимая сущность.
	**обобщение (generalization)** — отношение наследования или реализации интерфейса. Со стороны стрелки находится суперкласс или интерфейс.

На самом деле тут все очень просто. Последняя стрелочка фактически обозначает, что один класс наследуется от другого класса. А первая и вторая стрелочки – это то, что один объект хранит ссылку на второй объект. И это все.

Если ромбик у ссылки черный, то ссылка слабая: объекты могут существовать друг без друга. Если ромбик белый, то объекты связаны сильно, как, например, класс `HttpRequest` и его дочерний класс `HttpRequest.Builder`.

1.5 Список паттернов

Виды паттернов будем обозначать разными цветами и буквами:

`B` — поведенческие (behavioral);

`C` — порождающие (creational);

`S` — структурные (structural).

И наконец список из 23-х паттернов проектирования:

C — Абстрактная фабрика	S — Фасад	S — Прокси
S — Адаптер	C — Фабричный метод	B — Наблюдатель
S — Мост	S — Приспособленец	C — Одиночка
C — Строитель	B — Интерпретатор	B — Состояние
B — Цепочка обязанностей	B — Итератор	B — Стратегия
B — Команда	B — Посредник	B — Шаблонный метод
S — Компоновщик	B — Хранитель	B — Посетитель
S — Декоратор	C — Прототип	

‹ Предыдущая лекция

Следующая лекция ›

+22

Комментарии (3) популярные **новые** старые

JavaCoder

Введите текст комментария

Марат Гарипов Уровень 75 17 ноября 2022, 23:26

"Если ромбик у ссылки черный, то ссылка слабая: объекты могут существовать друг без друга. Если ромбик белый, то объекты связаны сильно"... а не наоборот?

Ответить +4

Oleg Khilko Уровень 51 15 августа 2022, 19:01

Пошарю первым, раз еще мало людей сюда дошло.✌

Большая шпаргалка по шаблонам проектирования

Вернулся добавить:

Так как дальше пойдет объяснение паттернов и с очень маленькой вероятностью вы реально врубитесь что это и как оно работает с первого раза, лучше посмотреть n авторов(прочитать n книг) и сформировать m нейронных связей и тогда вы действительно сможете отличить абстрактную фабрику от обычной.

Поэтому делюсь тем что нарыл на текущий момент сам:
Плейлист объяснения всех паттернов
+
Head First про Паттерны проектирования
+
Шаблоны проектирования "банды четырёх (GoF)"

Какой-то закономерности по тому что и когда именно читать я не вывел. В идеальном мире необходимо каждый паттерн добавлять по очереди в свой "рацион" и просто со временем вы попробуете их все. Ничего сложного в них нет.

Ответить +17

Mansur Ilmiev Уровень 22 24 июля 2022, 18:22

Хорошая статья.
Дало краткое понимание того, зачем же всё таки стоит изучать эти ваши "шаблоны" и какой с этого толк

Ответить

+3

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.