1 Введение

Шаблоны проектирования — это один из важнейших компонентов объектно- ориентированной технологии разработки программного обеспечения Они широко применяются в инструментах анализа, подробно описываются в книгах и часто обсуждаются на семинарах по объектно-ориентированному проектированию.

1.1 Предыстория

Много лет назад архитектор по имени Кристофер Александер задумался над вопросом: «Является ли качество объективной категорией?». Следует ли считать представление о красоте сугубо индивидуальным, или люди могут прийти к общему соглашению, согласно которому некоторые вещи будут считаться красивыми, а другие нет?

Александер размышлял о красоте с точки зрения архитектуры. Его интересовало, по каким показателям мы оцениваем архитектурные проекты. Например, если некто вознамерился спроектировать крыльцо дома, то как он может получить гарантии, что созданный им проект будет хорош? Можем ли мы знать заранее, что проект будет действительно хорош? Имеются ли объективные основания для вынесения такого суждения?

Александер принял как постулат, что в области архитектуры такое объективное основание существует. Суждение о том, что некоторое здание является красивым,— это не просто вопрос вкуса. Красоту можно описать с помощью объективных критериев, которые могут быть измерены.

К похожим выводам пришли и исследователи в области в культурологии. В пределах одной культуры большинство индивидуумов имеют схожие представления о том, что является сделанным хорошо и что является красивым. В основе их суждений есть нечто более общее, чем сугубо индивидуальные представления о красоте.

Исходная посылка создания шаблонов проектирования также состояла в необходимости объективной оценки качества программного обеспечения. Александер сформулировал для себя следующие вопросы:

1. Что есть такого в проекте хорошего качества, что отличает его от плохого проекта?
2. Что именно отличает проект низкого качества от проекта высокого качества?

Эти вопросы навели Александера на мысль о том, что если качество проекта является объективной категорией, то мы можем явно определить, что именно делает проекты хорошими, а что — плохими.

Александер изучал эту проблему, обследуя множество зданий, городов, улиц и всего прочего, что люди построили для своего проживания. В результате он обнаружил, что все, что было построено хорошо, имело между собой нечто общее. Архитектурные структуры отличаются друг от друга, даже если они относятся к одному и тому же типу. Однако, не взирая на имеющиеся различия, они могут оставаться высококачественными.

1.2 От архитектурных шаблонов к шаблонам проектирования программного обеспечения

Но какое отношение может иметь весь этот архитектурный материал к специали стам в области программного обеспечения, коими мы являемся? В начале 1990х некоторые из опытных разработчиков программного обеспе чения Gangs of Four (GOF) ознакомились с упоминавшейся выше работой Александера об архитектур ных шаблонах. Они задались вопросом, возможно ли применение идеи архитектурных шаблонов при реализации проектов в области создания программного обеспечения. Сформулируем те вопросы, на которые требовалось получить ответ.

- Существуют ли в области программного обеспечения проблемы, возникающие снова и снова, и могут ли они быть решены тем же способом?
- Возможно ли проектирование программного обеспечения в терминах шабло нов — т.е. создание конкретных решений на основе тех шаблонов, которые будут выявлены в поставленных задачах?

Интуиция подсказывала исследователям, что ответы на оба эти вопроса опреде ленно будут положительными. Следующим шагом необходимо было идентифициро вать несколько подобных шаблонов и разработать стандартные методы каталогиза ции новых. Хотя в начале 1990х над шаблонами проектирования работали многие исследова тели, наибольшее влияние на это сообщество оказала книга Гаммы, Хелма, Джонсона и Влиссайдеса Шаблоны проектирования: элементы многократного использования кода в объектноориентированном программировании. Эта работа приобрела очень широкую из вестность. Свидетельством ее популярности служит тот факт, что четыре автора кни ги получили шутливое прозвище "банда четырех". Важно понять, что авторы сами не создавали тех шаблонов, которые описаны в их книге. Скорее, они идентифицировали эти шаблоны как уже существующие в разра ботках, выполненных сообществом создателей программного обеспечения. Поэтому у некоторых разработчиков возникает вопрос, зачем изучать шаблоны?

1.3 Зачем нужно изучать шаблоны

Теперь, когда мы знаем, что такое шаблоны проектирования, можно попытаться ответить на вопрос, зачем нужно их изучать. На то имеется несколько причин, часть из которых вполне очевидна, тогда как об остальных этого не скажешь. Чаще всего причины, по которым следует изучать шаблоны проектирования, формулируют следующим образом.

- Возможность многократного использования. Повторное использование решений из уже завершенных успешных проектов позволяет быстро приступить к решению новых проблем и избежать типичных ошибок. Разработчик получает прямую выгоду от использования опыта других разработчиков, избежав необходимости вновь и вновь изобретать велосипед.

- Применение единой терминологии. Профессиональное общение и работа в группе (команде разработчиков) требует наличия единого базового словаря и единой точки зрения на проблему. Шаблоны проектирования предоставляют подобную общую точку зрения как на этапе анализа, так и при реализации проекта.

- Шаблоны проектирования предоставляют нам абстрактный высокоуровневый взгляд как на проблему, так и на весь процесс объектноориентированной разра ботки. Это помогает избежать излишней детализации на ранних стадиях проекти рования.

1.4 Другие преимущества

В результате применения шаблонов проектирования повышается эффективность труда отдельных исполнителей и всей группы в целом. Это происходит из-за того, что начинающие члены группы видят на примере более опытных разработчиков, как шаблоны проектирования могут применяться и какую пользу они приносят. Совместная работа дает новичкам стимул и реальную возможность быстрее изучить и освоить эти новые концепции. Применение многих шаблонов проектирования позволяет также создавать более модифицируемое и гибкое программное обеспечение. Причина состоит в том, что эти решения уже испытаны временем. Поэтому использование шаблонов позволяет создавать структуры, допускающие их модификацию в большей степени, чем это воз можно в случае решения, первым пришедшего на ум. Шаблоны проектирования, изученные должным образом, существенно помогают общему пониманию основных принципов объектноориентированного проектирова ния.

«Бандой четырех» было предложено несколько стратегий создания хорошего объ ектноориентированного проекта. В частности, они предложили следующее:

- Проектирование согласно интерфейсам.

- Предпочтение синтеза наследованию.

- Выявление изменяющихся величин и их инкапсуляция.

Эти стратегии использовались в большинстве шаблонов проектирования, обсуж даемых в данной книге. Для оценки полезности указанных стратегий вовсе не обяза тельно изучить большое количество шаблонов — достаточно всего нескольких. При обретенный опыт позволит применять новые концепции и к задачам собственных проектов, даже без непосредственного использования шаблонов проектирования.

Еще одно преимущество состоит в том, что шаблоны проектирования позволяют разработчику или группе разработчиков находить проектные решения для сложных проблем, не создавая громоздкой иерархии наследования классов.

2 Сруктурные паттерны

Структурные шаблоны связаны со способами формирования более крупных структур. Структурные шаблоны классов используют наследование или композицию для создания интерфейсов или реализаций.

2.1 Фасад

2.1.1 Назначение

В книге «банды четырех» назначение шаблона Facade (фасад) определяется следующим образом:

> Предоставление единого интерфейса для набора различных интерфейсов в системе. Шаблон Facade определяет интерфейс более высокого уровня, что упрощает работу с системой.

2.1.2 Мотивация

В основном этот шаблон используется в тех случаях, когда необходим новый способ взаимодействия с системой — более простой в сравнении с уже существующим. Кроме того, он может применяться, когда требуется использовать систему некоторым специфическим образом — например, обращаться к программе трехмерной графики для построения двухмерных изображений. В этом случае нам потребуется специальный метод взаимодействия с системой, поскольку будет использоваться лишь часть ее функциональных возможностей.

Примером фасада является JDBC интерфейс в Java, так как его пользователи создают подключение через интерфейс java.sql.Connection, реализается которого их не касается.

2.1.3 Использование

- Когда нет необходимости использовать все функциональные возможности сложной системы и можно создать новый класс, который будет содержать все необходимые средства доступа к базовой системе. Если предполагается работа лишь с ограниченным набором функций исходной системы, как это обычно и бывает, интерфейс (API), описанный в новом классе, будет намного проще, чем стандартный интерфейс, разработанный создателями основной системы.
- Используйте фасад для отделения подсистем от клиентов, способствуя тем самым снижению зависимостей и повышению переносимости
- Если подсистемы системы между собой зависимы, то, используя фасад, можно упростить зависимости между подсистемами.

2.2 Адаптер

Используется также другое название — обертка (wrapper).

2.2.1 Назначение

В книге «банды четырех» назначение шаблона определяется следующим образом:

> Преобразование интерфейса класса в интерфейс, нужный пользователю.

2.2.2 Мотивация

Рассмотрим, например, редактор чертежей, который позволяет пользователям рисовать и упорядочивать графические элементы (линии, многоугольники, текст и т. Д.) в виде изображений и диаграмм. Ключевой абстракцией редактора рисунков является графический объект, который имеет редактируемую форму и может рисовать сам. Интерфейс для графических объектов определяется абстрактным классом Shape. Редактор определяет подкласс Shape для каждого типа графического объекта: класс LineShape для линий, класс PolygonShape для полигонов и т. Д.

Классы элементарных геометрических фигур, таких как LineShape и PolygonShape, довольно просты в реализации, поскольку их возможности рисования и редактирования по своей природе ограничены. Но подкласс TextShape, который может отображать и редактировать текст, значительно сложнее реализовать, поскольку даже базовое редактирование текста включает в себя сложное обновление экрана и управление буфером. Между тем, готовый инструментарий пользовательского интерфейса уже может предоставлять сложный класс TextView, который реализует функции TextShape, но имеет другой интерфейс для отображения и редактирования текста. В идеале мы хотели бы повторно использовать TextView как подкласс Shape, но инструментарий не был разработан так, чтобы можно было использовать класс TextView. Поэтому мы не можем использовать объекты TextView и Shape взаимозаменяемо.

Как могут существующие и не связанные классы, такие как TextView, работать в приложении, которое ожидает классы с другим и несовместимым интерфейсом? Мы могли бы изменить класс TextView, чтобы он соответствовал интерфейсу Shape, но это невозможно, если у нас нет исходного кода. Даже если бы мы это сделали, не было бы смысла менять TextView, так как неоптимально менять код работающего класса для того, чтобы сделать возможным его применение в каком-то одном приложении.

Вместо этого мы можем определить TextShape, чтобы он адаптировал интерфейс TextView к Shape. Мы можем сделать это одним из двух способов:

1. путем наследования интерфейса Shape и реализации Text View;
2. путем создания экземпляра TextView внутри TextShape и реализации TextShape в терминах интерфейса Text View.

Эти два подхода соответствуют версии класса и объекта шаблона Adapter. Класс TextShape является адаптером.

2.2.3 Использование

- Вы хотите использовать существующий класс, и его интерфейс не соответствует тому, который вам нужен
- Вы хотите создать повторно используемый класс, который взаимодействует с несвязанными или непредвиденными классами, то есть классами, которые не обязательно имеют совместимые интерфейсы.
- Вам нужно использовать несколько существующих подклассов, не непрактично адаптировать их интерфейсы путем наследования каждого. Адаптер может адаптировать интерфейс их родителя.