**Ответы на вопросы по паттернам проектирования**

1. **Почему до 1991 года никто не выпускал серьёзных работ по шаблонированию разработки?**

На самом деле шаблоны существовали всегда, просто воспринимались они не так серьезно.

1. **В следствии чего появляются новые паттерны?**

Здесь все очевидно, наверняка архитектура вашего проекта уже имела какую-нибудь реализацию или вы строите свое приложение каким-то определенным образом, согласно стандарту.

1. **Как ты считаешь, нужно ли использовать паттерны или нет?**

На мой взгляд, нужно искать золотую середину. И скорее да, чем нет.

1. **Какие шаблоны проектирования самые популярные? Какие спрашивают на собеседованиях? :) Какие паттерны, на твой взгляд, самые полезные?**

Порождающие шаблоны

Фабрика, Абстрактная фабрика, и связке с Синглтон Прототип и Строитель.

1. **Плюсы и минусы паттерна неизменяемого объекта (immutable).**

Очень мало инфы. Однако, кое-где написано, что благодаря нему видна более четкая логика и потокобезопасность тк неизменяемый.

1. **Говорят, что паттернами увлечены, в основном, программисты среднего уровня, опытные относятся к ним куда прохладней. Почему так?**

Слегка задели эту тему в докладе. Есть 2 подхода ко всему этому. С одной стороны прикольно, когда у вас есть шаблон, который популярен, который используют все. Да и на подсознательном уровне — о это уже кто-то сделал, сделаю и я. Однако не все так здорово, не всегда это работает. Иногда возникают ситуации, когда вам приходится подстраивать свою архитектуру под это, а потом, когда вам нужно как-то расширить свою архитектуру — то все. Ваш проект обрастает ненужным кодом или увеличивается уровень абстракции...

1. **Когда лучше использовать MVC, а когда MVVM?**

MVC читается классическим вариантом. модель приложения, [пользовательский интерфейс](https://ru.wikipedia.org/wiki/Пользовательский_интерфейс) и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

MVVM удобно использовать вместо классического [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller)и ему подобных в тех случаях, когда в платформе, на которой ведётся разработка, присутствует связывание данных(это процесс, который устанавливает соединение между [UI](https://ru.wikipedia.org/wiki/UI) (пользовательским интерфейсом) приложения и [бизнес-логикой](https://ru.wikipedia.org/wiki/Бизнес-логика). Если настройки и уведомления установлены правильно, данные отражают изменения, когда они сделаны. Это может также значить, что когда UI изменяется, лежащие в его основе данные будут отражать эти изменения.)

1. **В чем различия антипаттернов "Спагетти-код", "Лазнья-код", "Равиоли-код"?**

[Спагетти-код](https://ru.wikipedia.org/wiki/Спагетти-код) (Spaghetti code, иногда «макароны»): Код с чрезмерно запутанным порядком выполнения

**Спагетти-код** — плохо спроектированная, слабо структурированная, запутанная и трудная для понимания [программа](https://ru.wikipedia.org/wiki/Компьютерная_программа), особенно содержащая много операторов[GOTO](https://ru.wikipedia.org/wiki/GOTO) (особенно переходов назад), [исключений](https://ru.wikipedia.org/wiki/Обработка_исключений) и других конструкций, ухудшающих [структурированность](https://ru.wikipedia.org/wiki/Структурное_программирование)[[1]](https://ru.wikipedia.org/wiki/Спагетти-код" \l "cite_note-AntiPatterns-1). Самый распространённый [антипаттерн](https://ru.wikipedia.org/wiki/Антипаттерн)программирования. Остальные необщепринятые.

* [Лазанья-код](https://ru.wikipedia.org/w/index.php?title=Лазанья-код&action=edit&redlink=1) (Lasagnia code, или "лук" (onion)): Использование неоправданно большого количества уровней абстракции
* [Равиоли-код](https://ru.wikipedia.org/w/index.php?title=Равиоли-код&action=edit&redlink=1) (Ravioli code, или "пельмени"): Объекты настолько "склеены" между собой, что практически не допускают рефакторинга

1. **Расскажите, пожалуйста, подробнее про антипаттерн "Золушкина туфелька "**

**Попытка "натянуть" на объект уже имеющийся малоподходящий по смыслу интерфейс, вместо создания нового.**

1. **Пользовались ли мы на практике каким-нибудь шаблонами проектирования?**
2. **Имеет ли шаблон concurrency какое-то отношение к многопоточности?**

Самое прямое отношение.

1. **Что такое антипаттерны, и для чего они применяются?**

это классы наиболее часто внедряемых плохих решений проблем. Они изучаются, как категория, в случае когда их хотят избежать в будущем, и некоторые отдельные случаи их могут быть распознаны при изучении неработающих систем.

1. **Какие книги вы посоветуете для обязательного прочтения на тему паттернов проектирования?**

Все рекомендуют «Паттерны проектирования» (Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс)

1. **Считаете ли вы, что знание и следование патернам проектирования является необходимым для современного специалиста?**

Да, на мой вгляд это необходимо

1. **Следуете ли вы паттернам?**

Странный впрос

1. **Паттернов много. А какой базис в бы выделили?**

Классный вопрос.

1. **Почему примитивы синхронизации указывались как паттерны в ответвлении concurrency?**

А почему бы и нет. Используются для более эффективного написания [многопоточных](https://ru.wikipedia.org/wiki/Многопоточность) программ, и предоставляет готовые решения проблем [синхронизации](https://ru.wikipedia.org/wiki/Синхронизация_(информатика)).

1. **Какие существуют паттерны для взаимодействия с БД?**
   * + 1. [4.2.3.1 Активная запись (Active Record)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.1)
       2. [4.2.3.2 Единица работы (Unit Of Work)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.2)
       3. [4.2.3.3 Загрузка по требованию (Lazy Load)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.3)
       4. [4.2.3.4 Коллекция обьектов (Identity Map)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.4)
       5. [4.2.3.5 Множество записей (Record Set)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.5)
       6. [4.2.3.6 Наследование с одной таблицей (Single Table Inheritance)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.6)
       7. [4.2.3.7 Наследование с таблицами для каждого класса (Class Table Inheritance)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.7)
       8. [4.2.3.8 Оптимистическая автономная блокировка (Optimistic Offline Lock)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.8)
       9. [4.2.3.9 Отображение с помощью внешних ключей](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.9)
       10. [4.2.3.10 Отображение с помощью таблицы ассоциаций (Association Table Mapping)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.10)
       11. [4.2.3.11 Пессимистическая автономная блокировка (Pessimistic Offline Lock)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.11)
       12. [4.2.3.12 Поле идентификации (Identity Field)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.12)
       13. [4.2.3.13 Преобразователь данных (Data Mapper)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.13)
       14. [4.2.3.14 Cохранение сеанса на стороне клиента (Client Session State)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.14)
       15. [4.2.3.15 Cохранение сеанса на стороне сервера (Server Session State)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.15)
       16. [4.2.3.16 Шлюз записи данных (Row Data Gateway)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.16)
       17. [4.2.3.17 Шлюз таблицы данных (Table Data Gateway)](http://citforum.ru/SE/project/pattern/p_3.shtml" \l "4.2.3.17)

**Еще DAO и DTO**

1. **Как нужно выбирать паттерн?**

На самом деле с выбором паттерна никаких проблем. Каждый паттерн решает какую-то задачу. Вам остается определится с тем, что бы вы хотели.

1. **Что такое блоб?**

**Блоб**(от [англ.](https://ru.wikipedia.org/wiki/Английский_язык)***b****inary****l****inked****ob****ject***—** объект двоичной компоновки) — [объектный файл](https://ru.wikipedia.org/wiki/Объектный_файл)без публично доступных исходных кодов, [загружаемый](https://ru.wikipedia.org/wiki/Компоновщик)в ядро[операционной системы](https://ru.wikipedia.org/wiki/Операционная_система). Обычно этот термин применяется только по отношению к модулям, загружаемым в [ядро](https://ru.wikipedia.org/wiki/Ядро_операционной_системы)[свободной](https://ru.wikipedia.org/wiki/Свободное_программное_обеспечение)или [открытой](https://ru.wikipedia.org/wiki/Открытое_программное_обеспечение)операционной системы; термин редко применяется по отношению к коду, выполняющемуся не в режиме ядра, например, код [BIOS](https://ru.wikipedia.org/wiki/BIOS), [микропрограммный код устройств](https://ru.wikipedia.org/wiki/Встроенное_программное_обеспечение), программы, выполняющиеся в пользовательском режиме.

1. **Как бороться с адом зависимостей?**

**DLL hell** (DLL-кошмар, буквально: [DLL](https://ru.wikipedia.org/wiki/DLL)-ад) — тупиковая ситуация, связанная с управлением динамическими библиотеками [DLL](https://ru.wikipedia.org/wiki/DLL) в операционной системе[Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows).

Данные меры рекомендуют предпринимать одновременно для получения наилучшего результата:

* подсчитать контрольную сумму кода функции, вызываемой из DLL - сравнить с контрольной суммой функции, используемой при написании программы.
* Операционная система должна поставляться совместно с менеджером пакетов, чтобы иметь возможность прослеживать все взаимозависимости DLL, при этом использование менеджера пакетов должно поощряться, а индивидуальная инсталляция DLL — по возможности отвергаться.
* Централизованное распространение библиотек
* Допустить возможность параллельного использования нескольких версий одной и той же DLL [[1]](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sbscs/setup/side_by_side_assemblies.asp).
* При модификации программного обеспечения для частного использования поставлять также модифицированные версии DLL.
* Во время проектирования DLL должна тщательно продумываться концепция функций и версий.
* DLL не должны использоваться без необходимости, а библиотеки, связанные только с одним приложением, должны подключаться статически (в EXE-файл).

**Часть вторая паттерны**

**1) Мост**

3.1.6 Мост (Bridge), Handle (описатель) или Тело (Body) - GoF

|  |  |
| --- | --- |
| **Проблема** | Требуется отделить абстракцию от реализации так, чтобы и то и другое можно было изменять независимо. При использовании наследования реализация жестко привязывается к абстракции, что затрудняет независимую модификацию. |
| **Решение** | Поместить абстракцию и реализацию в отдельные иерархии классов. |
| **Рекомендации** | Можно использовать если, например, реализацию необходимо выполнять во время реализации программы. |
| **Пример** | "Абстракция" определяет интерфейс "Абстракции" и хранит ссылку на объект "Реализация", "УточненнаяАбстракция" расширяет интерфейс, определенный "Абстракцией". "Реализация" определяет интерфейс для классов реализации, он не обязан точно соответствовать интерфейсу класса "Абстракция" - оба интерфейса могут быть совершенно различны. Обычно интерфецйс класса "Реализация" предоставляет только примитивные операции, а класс "Абстракция" определяет операции более высокого уровня, базирующиеся на этих примитивных. "КонкретнаяРеализация" содержит конкретную реализацию класса "Реализация". Объект "Абстракция" перенаправляет своему объекту "Реализация" запросы "Клиента". |
| **Преимущества** | Отделение реализации от интерфейса, то есть, "Реализацию" "Абстракции" можно конфигурировать во время выполнения. Кроме того, следует упомянуть, что разделение классов "Абстракция" и "Реализация" устраняет зависимости от реализации, устанавливаемые на этапе компиляции: чтобы изменить класс "Реализация" вовсе не обязательно перекомпилировать класс "Абстракция". |

**2) Строитель**

3.3.5 Строитель (Builder) - GoF

|  |  |
| --- | --- |
| **Проблема** | Отделить конструирование сложного объекта от его представления, так чтобы в результате одного и того же конструирования могли получаться различные представления. Алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой. |
| **Решение** | "Клиент" создает объект - распорядитель "Директор" и конфигурирует его объектом - "Строителем". "Директор" уведомляет "Строителя" о том, что нужно построить очередную часть "Продукта". "Строитель" обрабатывает запросы "Директора" и добавляет новые части к "Продукту", затем "Клиент" забирает "Продукт" у "Строителя". |
| **Преимущества** | Объект "Строитель" предоставляет объекту "Директор" абстрактный интерфейс для конструирования "Продукта", за которым может скрыть представление и внутреннюю структуру продукта, и , кроме того, процесс сборки "продукта". Для изменения внутреннего представления "Продукта" достаточно определить новый вид "Строителя". Данный паттерн изолирует код, реализующий создание объекта и его представление. |

**3) Абстрактная фабрика**

3.3.1 Абстрактная фабрика (Abstract Factory, Factory), др. название Инструментарий (Kit) - GoF

|  |  |
| --- | --- |
| **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. |

**4)Адаптер**

3.1.1 Адаптер (Adapter) - GoF

|  |  |
| --- | --- |
| **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. |

**5)Декоратор(Обертка)**

3.1.2 Декоратор (Decorator) или Оболочка (Wrapper) - GoF

|  |  |
| --- | --- |
| **Проблема** | Возложить дополнительные обязанности (прозрачные для клиентов) на отдельный объект, а не на класс в целом. |
| **Рекомендации** | Применение нескольких "Декораторов" к одному "Компоненту" позволяет произвольным образом сочетать обязанности, например, одно свойство можно добавить дважды. |
| **Решение** | Динамически добавить объекту новые обязанности не прибегая при этом к порождению подклассов (наследованию). "Компонент"определяет интерфейс для обьектов, на которые могут быть динамически возложены дополнительные обязанности, "КонкретныйКомпонент" определяет объект, на который возлагаются дополнительные обязанности, "Декоратор" - хранит ссылку на объект "Компонент" и определяет интерфейс, соответствующий интерфейсу "Компонента". "КонкретныйДекоратор" возлагает дополнительные обязанности на компонент. "Декоратор" переадресует запросы объекту "Компонент". |
| **Преимущества** | Большая гибкость, чем у статического наследования: можно добавлять и удалять обязанности во время выполнения программы в то время как при использовании наследования надо было бы создавать новый класс для каждой дополнительной обязанности. Данный паттерн позволяет избежать перегруженных методами классов на верхних уровнях иерархии - новые обязанности можно добавлять по мере необходимости. |
| **Недостатки** | "Декоратор" и его "Компонент" не идентичны, и, кроме того, получается что система состоит из большого числа мелких обьектов, которые похожи друг на друга и различаются только способом взаимосвязи а не классом и не значениями своих внутренних переменных - такая система сложна в изучении и отладке. |

**6)Синглтон**

3.3.2 Одиночка (Singleton) - GoF

|  |  |
| --- | --- |
| **Проблема** | Какой специальный класс должен создавать "Абстрактную фабрику", см. [3.3.1](http://citforum.ru/SE/project/pattern/p_2.shtml" \l "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. |