DESIGN PATTERNS INTERVIEW JAVA DEVELOPER (ADVANCED)

Ответы на вопросы на собеседование Design patterns.

Vasyl K 18:10:00 Добавить комментарий

- **ЧТО ТАКОЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ?**

Шаблоны проектирования GoF – это многократно используемые решения широко распространенных проблем, возникающих при разработке программного обеспечения.

- **ИЗ КАКИХ ЭЛЕМЕНТОВ СОСТОИТ ШАБЛОН?**

В общем случае шаблон состоит из четырех основных элементов:

1. имя. Точное имя предоставляет возможность сразу понять проблему и определить решение. Уровень абстракции при проектировании повышается;
2. задача. Область применения в рамках решения конкретной проблемы;
3. решение. Абстрактное описание элементов дизайна задачи проектирования и способа ее решения с помощью обобщенного набора классов;
4. результаты.

- **КАКИЕ ЕСТЬ ТИПЫ ШАБЛОНОВ?**

Выделяются порождающие шаблоны, структурные шаблоны и шаблоны поведения а также антышаблоны.

- **НАЗОВИТЕ ПОРОЖДАЮЩИЕ ШАБЛОНЫ, И КРАТКО ОПИШИТЕ ИХ.**

Порождающие шаблоны предназначаются для организации процесса создания объектов и все до единого соответствуют шаблону Creator из GRASP.

К порождающим шаблонам относятся:

- Abstract Factory (Абстрактная Фабрика) – предоставляет интерфейс для создания связанных между собой объектов семейств классов без указания их конкретных реализаций (families of product objects);
- Factory Method (Фабричный метод) – определяет интерфейс для создания объектов из иерархического семейства классов на основе передаваемых данных (subclass of object that is instantiated);
- Builder (Строитель) – создает объект конкретного класса различными способами (how a composite object gets created);

- Singleton (Одиночка) – гарантирует существование только одного или конечного числа экземпляров класса (the sole instance of a class);
- Prototype (Прототип) – применяется при создании сложных объектов. На основе прототипа объекты сохраняются и воссоздаются, например, путем копирования (class of object that is instantiated).

- **НАЗОВИТЕ ШАБЛОНЫ ПОВЕДЕНИЯ, И КРАТКО ОПИШИТЕ ИХ.**

Шаблоны поведения GoF характеризуют способы взаимодействия классов или объектов между собой.

К шаблонам поведения относятся:

- Chain of Responsibility (Цепочка Обязанностей) – организует независимую от объекта-отправителя цепочку не знающих возможностей друг друга объектов-получателей, которые передают запрос друг другу (object that can fulfill a request);
- Command (Команда) – используется для определения по некоторому признаку объекта конкретного класса, которому будет передан запрос для обработки (when and how a request is fulfilled);
- Iterator (Итератор) – позволяет последовательно обойти все элементы коллекции или другого составного объекта, не зная деталей внутреннего представления данных (how an aggregate's elements are accessed, traversed);
- Mediator (Посредник) – позволяет снизить число связей между классами при большом их количестве, выделяя один класс, знающий все о методах других классов (how and which objects interact with each other);
- Memento (Хранитель) – сохраняет текущее состояние объекта для дальнейшего восстановления (what private information is stored outside an object, and when);
- Observer (Наблюдатель) – позволяет при зависимости между объектами типа «один ко многим» отслеживать изменения объекта (number of objects that depend on another object; how the dependent objects stay up to date);
- State (Состояние) – позволяет объекту изменять свое поведение за счет изменения внутреннего объекта состояния (states of an object);
- Strategy (Стратегия) – задает набор алгоритмов с возможностью выбора одного из классов для выполнения конкретной задачи во время создания объекта (an algorithm);
- Template Method (Шаблонный Метод) – создает родительский класс, использующий несколько методов, реализация которых возложена на производные классы (steps of an algorithm);
- Visitor (Посетитель) – представляет операцию в одном или нескольких связанных классах некоторой структуры, которую вызывает специфичный для каждого такого класса метод в другом классе (operations that can be applied to object(s) without changing their class(es));
- Interpreter (Интерпретатор) – для определенного способа представления информации определяет правила (grammar and interpretation of a language).

- **НАЗОВИТЕ СТРУКТУРНЫЕ ШАБЛОНЫ, И КРАТКО ОПИШИТЕ ИХ.**

Структурные шаблоны GoF отвечают за композицию объектов и классов, и не только за объединение частей приложения, но и за их разделение.

К структурным шаблонам относятся:

- Adapter (Адаптер) – применяется при необходимости использовать классы вместе с несвязанными интерфейсами. Поведение адаптируемого класса при этом изменяется на необходимое (interface to an object);

- Bridge (Мост) – разделяет представление класса и его реализацию, позволяя независимо изменять то и другое (implementation of an object);
- Composite (Компоновщик) – группирует объекты в иерархические древовидные структуры и позволяет работать с единичным объектом так же, как с группой объектов (structure and composition of an object);
- Decorator (Декоратор) – представляет способ изменения поведения объекта без создания подклассов. Позволяет использовать поведение одного объекта в другом (responsibilities of an object without subclassing);
- Facade (Фасад) – создает класс с общим интерфейсом высокого уровня к некоторому числу интерфейсов в подсистеме (interface to a subsystem);
- Flyweight (Легковес) – разделяет свойства класса для оптимальной поддержки большого числа мелких объектов (storage costs of objects);
- Proxy (Заместитель) – подменяет сложный объект более простым и осуществляет контроль доступа к нему (how an object is accessed… its location).

- **КАКИЕ АНТИШАБЛОНЫ ВЫ ЗНАЕТЕ?**

Некоторые антишаблоны:

- Big ball of mud (Большой Ком Грязи) – термин для системы или просто программы, которая не имеет хоть немного различимой архитектуры. Как правило, включает в себя более одного антишаблона. Этим страдают системы, разработанные людьми без подготовки в области архитектуры ПО.
- Yo-Yo problem (Проблема Йо-Йо) – возникает, когда необходимо разобраться в программе, иерархия наследования и вложенность вызовов методов которой очень длинны и сложны. Программисту вследствие этого необходимо лавировать между множеством различных классов и методов, чтобы контролировать поведение программы. Термин происходит от названия игрушки йо-йо.
- Magic Button – возникает, когда код обработки формы сконцентрирован в одном месте и, естественно, никак не структурирован.
- Magic Number – наличие в коде многократно повторяющихся одинаковых чисел или чисел, объяснение происхождения которых отсутствует.
- Gas Factory (Газовый Завод) – необязательный сложный дизайн для простой задачи.
- Analiys paralisys. В разработке ПО (Паралич анализа) – проявляет себя через чрезвычайно длинные фазы планирования проекта, сбора необходимых для этого артефактов, программного моделирования и дизайна, которые не имеют особого смысла для достижения итоговой цели.
- Interface Bloat (Распухший Интерфейс) – термин, используемый для описания интерфейсов, которые пытаются вместить в себя все возможные операции над данными.
- Accidental complexity (Случайная сложность) – проблема в программировании, которой легко можно было избежать. Возникает вследствие неправильного понимания проблемы или неэффективного планирования.

И другие…

- **ЧТО ТАКОЕ OOAD?**

OOAD, Object Oriented Analysis and Design (Объектно-ориентированный анализ и проектирование) – дисциплина, описывающая способы (варианты) задания (определения) объектов и их взаимодействие для решения проблемы, которая определена и описана в ходе объектно-ориентированного анализа.

Основная идея объектно-ориентированного анализа и проектирования (object-

oriented analysis and design) состоит в рассмотрении предметной области и логического решения задачи с точки зрения объектов (понятий и сущностей). В процессе объектно-ориентированного анализа основное внимание уделяется определению и описанию объектов (или понятий) в терминах предметной области. В процессе объектно-ориентированного проектирования определяются логические программные объекты, которые будут реализованы средствами объектно-ориентированного языка программирования. Эти программные объекты включают в себя атрибуты и методы. И, наконец, в процессе конструирования (construction) или объектно-ориентированного программирования (object-oriented programming) обеспечивается реализация разработанных компонентов и классов.

- **ЧТО ТАКОЕ OOD?**

OOD, Object Oriented Design (Объектно-ориентированное проектирование) – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

- **ЧТО ТАКОЕ OOA?**

OOA, Object Oriented Analysis (Объектно-ориентированный анализ) – это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области, это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

- **ЧТО ТАКОЕ DRY PRINCIPLES?**

DRY, Don't repeat yourself (не повторяй себя) – этот принцип настолько важен, что не требует повторения! Это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода, особенно в системах со множеством слоёв абстрагирования, простими словами НЕ пишите повторяющегося кода, используйте принцип абстракции, обобщая простые вещи в одном месте.

- **ЧТО ТАКОЕ KISS?**

KISS, Keep it short and simple или Keep it simple, stupid (делайте вещи проще) – это принцип проектирования и программирования, запрещающий использование более сложных средств, чем необходимо. Принцип декларирует простоту системы в качестве основной цели и/или ценности.

- **ЧТО ТАКОЕ YAGNI?**

YAGNI, You ain't gonna need it (Вам это не понадобится) – процесс и принцип проектирования ПО, при котором в качестве основной цели и/или ценности декларируется отказ от избыточной функциональности. Суть в том, чтобы реализовать только поставленные задачи и отказаться от избыточного функционала.

- **ЧТО ТАКОЕ YODA CONDITIONS?**

Yoda conditions (Условия Йоды в жаргоне программистов) – "безопасный" стиль записи выражений сравнения при программировании на языках с Си-синтаксисом, заключающийся в написании константного члена выражения (константы или вызова функции) слева от оператора сравнения (то есть 5 == a вместо привычного a == 5).
Такой стиль призван предотвратить свойственную данным языкам ошибку – использование оператора присваивания "=" вместо сравнения "==". Ошибочное использование присваивания превращает нотацию Йоды в попытку изменить константу, вызывая ошибку на этапе компиляции, что исключает возможность появления в готовой программе ошибок данного вида, а также облегчает их поиск и исправление в новом коде.

- **ЧТО ТАКОЕ CRC CARDS?**

CRC cards, Class-responsibility-collaboration card (Класс-Ответственность-Кооперация) – метод мозгового штурма, предназначенный для проектирования объектно-ориентированного программного обеспечения. Как правило, CRC-карты используются в тех случаях, когда сначала в процессе проектирования ПО определяются классы и способы их взаимодействий.
CRC-карты акцентируют внимание дизайнера на сущности класса и скрывают от него детали, рассмотрение которых на данном этапе будет контрпродуктивным. CRC-карты также заставляют дизайнера воздержаться от назначения классу слишком многих обязанностей.

- **ЧТО ТАКОЕ SOLID?**

SOLID, (single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion) – акроним для первых пяти принципов, которые означали пять основных принципов объектно-ориентированного программирования и проектирования. Эти принципы, когда применяются вместе, предназначены для повышения вероятности того, что программист создаст систему, которую будет легко поддерживать и расширять в течение долгого времени. Принципы SOLID – это руководства, которые могут применяться во время работы над программным обеспечением для удаления "кода с запашком" предписывая программисту выполнять рефакторинг исходного кода, пока тот не станет разборчиво написанным и расширяемым. Это часть общей стратегии гибкой и адаптивной разработки.

- **ЧТО ТАКОЕ SINGLE RESPONSIBILITY PRINCIPLE?**

Single responsibility principle – принцип единственной обязанности (на каждый класс должна быть возложена одна-единственная обязанность). Если один класс java реализует 2 набора функций, их сцепление создает ситуацию, при которой изменение одного нарушит имеющееся сочетание.

- **ЧТО ТАКОЕ OPEN/CLOSED PRINCIPLE?**

Open/closed principle – принцип объектно-ориентированного программирования, устанавливающий следующее положение: "программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения"; это означает, что такие сущности могут позволять менять свое поведение без изменения их исходного кода.

- **ЧТО ТАКОЕ LISKOV SUBSTITUTION PRINCIPLE?**

Liskov substitution principle – принцип подстановки Барбары Лисков (функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом. Подклассы не могут замещать поведения базовых классов. Подтипы должны дополнять базовые типы).

- **ЧТО ТАКОЕ INTERFACE SEGREGATION PRINCIPLE?**

Interface segregation principle – принцип разделения интерфейса (много специализированных интерфейсов лучше, чем один универсальный). Иными словами большие, объемные интерфейсы надо разбивать на мелкие таким образом, чтобы клиенты маленьких интерфейсов знали только о тех методах которые необходимы им в работе. И чтобы при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

- **ЧТО ТАКОЕ DEPENDENCY INVERSION PRINCIPLE?**

Dependency inversion principle – принцип инверсии зависимостей (зависимости внутри системы строятся на основе абстракций. Модули верхнего уровня не зависят от модулей нижнего уровня. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций).

- **ЧТО ТАКОЕ GRASP?**

GRASP, General Responsibility Assignment Software Patterns (общие шаблоны распределения обязанностей) – шаблоны проектирования, используемые для решения общих задач по назначению обязанностей классам и объектам. Известно девять GRAPS шаблонов.

- **КРАТКО ОПИШИТЕ ШАБЛОНЫ GRAPS.**

GRASP выделяет следующие 9 принципов-шаблонов:

- Information Expert (Информационные эксперт) - информационный эксперт описывает основополагающие принципы назначения обязанностей классам и объектам. Согласно описанию, информационным экспертом (объектом наделенным некоторыми обязанностями) является объект, обладающий максимумом информацией, необходимой для выполнения назначенных обязанностей.
- Creator (Создатель) - суть ответственности такого объекта в том, что он создает другие объекты. Сразу напрашивается аналогия с фабриками. Так оно и есть. Фабрики тоже имеют именно ответственность - Создатель.
- Controller (Контроллер) - отвечает за обработку входных системных событий, делегируя обязанности по их обработке компетентным классам. В общем случае, контроллер реализует один или несколько вариантов использования. Использование контроллеров позволяет отделить логику от представления, тем самым повышая возможность повторного использования кода.
- Low Coupling (Слабая связанность) - если объекты в приложении сильно связаны, то любой их изменение приводит к изменениям во всех связанных объектах. А это неудобно и порождает баги. Вот по-этому везде пишут, что необходимо, чтобы код был слабо связан и зависел только от абстракций.
- High Cohesion (Высокая сцепленность) - этот принцип тесно соотносится с слабой связанностью, и они идут в паре, когда одно всегда приводит к другому, это как мера того, что мы не нарушаем single resposibility principle. Вернее сказать, высокая сцепленность получается в результате соблюдения такого принципа из SOLID, как single resposibility principle (SRP).
- Pure Fabrication (Чистая выдумка или чистое синтезирование) - это класс, не отражающий никакого реального объекта предметной области, но специально придуманный для усиления связности, ослабления связанности или увеличения степени повторного использования. Pure Fabrication отражает концепцию сервисов в модели Программирование от предметной области.
- Indirection (Посредник) - шаблон перенаправление реализует низкую связность между классами, путем назначения обязанностей по их взаимодействию дополнительному объекту - посреднику.
- Protected Variations (Сокрытие реализации или защищенные изменения) - защищает элементы от изменения других элементов (объектов или подсистем) с помощью вынесения взаимодействия в фиксированный интерфейс. Всё взвимодействие между элементами должно происходить через него. Поведение может варьироваться лишь с помощью создания другой реализации интерфейса.
- Polymorphism (Полиморфизм) - позволяет обрабатывать альтернативные варианты поведения на основе типа и заменять подключаемые компоненты системы. Обязанности распределяются для различных вариантов поведения с помощью полиморфных операций для этого класса. Все альтернативные реализации приводятся к общему интерфейсу.