GRASP паттерны

Докладчик: Битюгов Алексей


ООП
Инкапсуляция
Абстракция
Наследование
Полиморфизм
Напряженный мыслительный процесс
GRASP паттерны

GRASP (общие шаблоны распределения ответственностей) были впервые опубликованы by Craig Larman в 1997 году в книге “Applying UML and Patterns”

Doing responsibilities:

1. Собственные действия объекта
2. Инициирование действий другого объекта
3. Контроль и координация активности других объектов

Knowing responsibilities:

1. Доступ к приватным данным
2. Знание о других объектах
3. Возможность получить некоторую информацию посредством вычислений

GRASP паттернов всего 9. В презентации они сгруппированы следующим образом:

- Общие принципы проектирования классов
  - High cohesion
  - Information expert
- Общие принципы построения связей между объектами
  - Low coupling
  - Indirection
  - Protected variations
  - Polymorphism
  - Pure fabrication
- Некоторые конкретные принципы
  - Creator
  - Controller

1. High cohesion / Высокая связность

Каждый класс должен быть предназначен для решения небольшой группы задач в рамках единой узкой специализации

+ Снижается зависимость от изменений
+ Легко переиспользовать
+ Понятно поведение
+ При решении задач программист взаимодействует лишь с небольшой ограниченной частью системы

Типы связности (от худшего к лучшему)

1. Случайная - группируются не связанные друг с другом элементы
2. Логическая - группируются элементы, выполняющие логически схожие функции
3. Временная - группируются элементы, использующиеся в общей временной фазе
4. Процедурная - группируются элементы, использующиеся в общей процедуре
5. Коммуникационная - группируются элементы, использующие общие данные
6. Последовательная - группируются элементы, для которых выходные данные одних будут входными данными других
7. Функциональная - группируются элементы, необходимые для выполнения общей задачи

✗ Не очень high cohesion

```
[Table("Employee", Schema = "dbo")]
2 references
public class Employee
{
    [Column("ID", Order = 1)]
    0 references
    public int? EmployeeId { get; set; }
    [Column("Name", Order = 2, TypeName = "Varchar(100)")]
    0 references
    public string? EmployeeName { get; set; }

    [InverseProperty("PrimaryEmployees")]
    0 references
    public DepartmentMaster? PrimaryDepartment { get; set; }

    [InverseProperty("SecondaryEmployees")]
    0 references
    public DepartmentMaster? SecondaryDepartment { get; set; }
}
```

✓ **Достаточно high cohesion**

```
0 references
internal class BlogEntityTypeConfiguration : IEntityTypeConfiguration<Blog>
{
    0 references
    public void Configure(EntityTypeBuilder<Blog> builder)
    {
        builder.ToTable("Blogs");
        builder.HasKey(x => x.Id);

        builder.Property(x => x.Id).HasColumnName("Id").ValueGeneratedOnAdd();
        builder.Property(x => x.Title).HasColumnName("Title").HasMaxLength(256).IsRequired();
        builder.Property(x => x.Content).HasColumnName("Content");
    }
}
```

Комментарий

High cohesion основывается на ООП и на принципе инкапсуляции – реализуем систему, состоящую из объектов, “разумно” группируя отдельные функции и данные, необходимые для их работы

Немного более абстрактный, чем все остальные паттерны

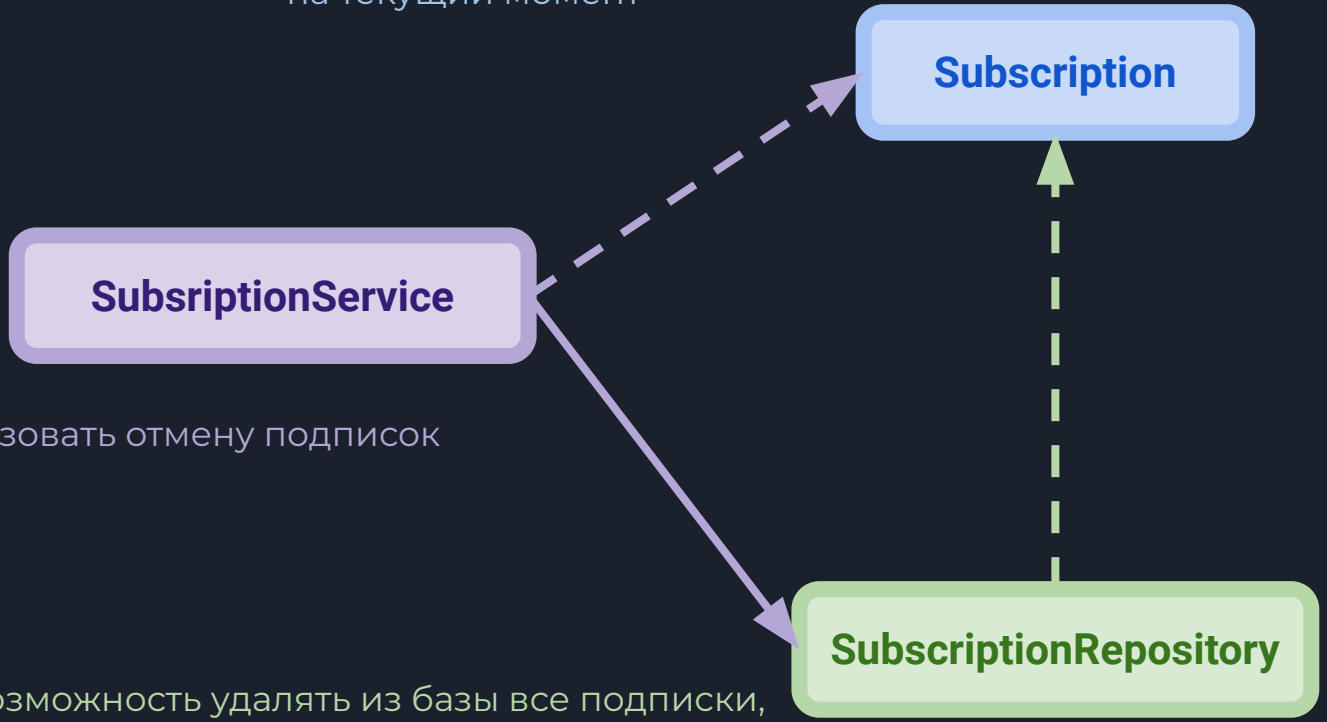
Похожие принципы: SRP и ISP из SOLID

2. Information Expert / Информационный эксперт

Ответственность по решению задачи должна назначаться классу, обладающим наибольшим количеством информации, необходимой для решения данной задачи

+ Не дублируется код
+ Не нужно дополнительно передавать данные между объектами

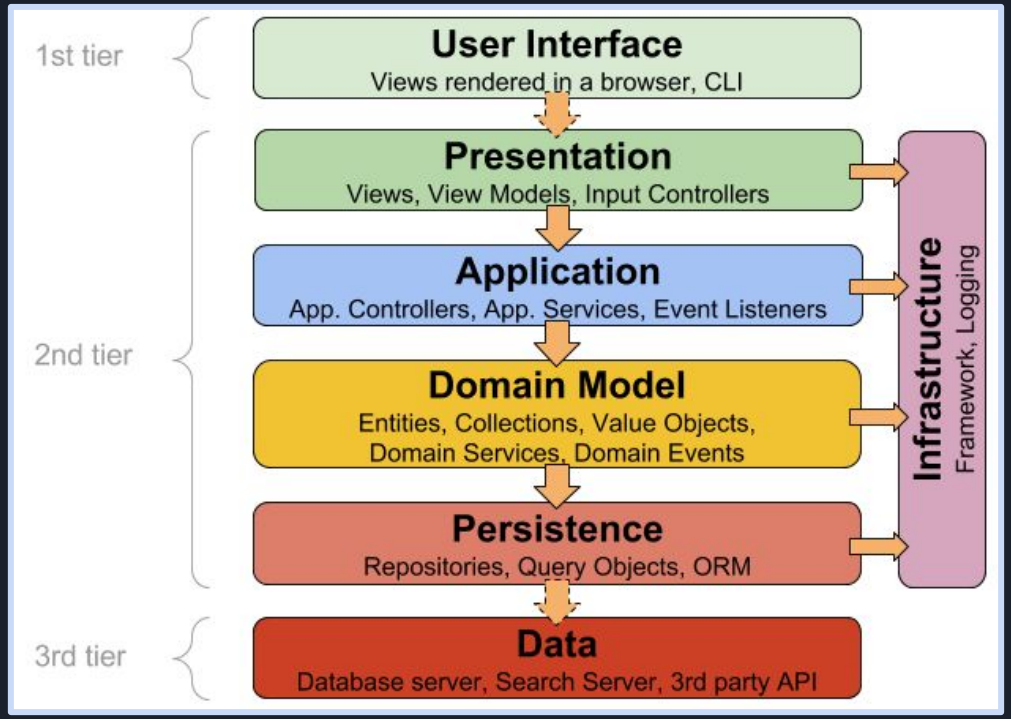
Реализовать проверку того, что подписка активна
на текущий момент



Реализовать отмену подписок

Добавить возможность удалять из базы все подписки,
соответствующие некоторому фильтру

Пример многослойной архитектуры



Комментарий

Information expert основывается на принципе инкапсуляции – так как данные хранятся объектами в закрытом виде, для решения задач нужно выбирать те из них, которые обладают необходимыми данными

Хорошо работает в сочетании с High cohesion (легко найти “эксперта”)

3. Low coupling / Слабое зацепление

Объекты должны иметь как можно меньше связей между собой, как можно меньше зависеть друг от друга или знать друг о друге

+ Легко изменять классы
+ Легко переиспользовать классы
+ Легко перестраивать архитектуру

Типы зацепления

1. По общей окружающей области - объекты используют общую область данных
2. По содержимому - один объект содержит другой объект
3. По управлению - один объект изменяет другой объект
4. По данным - один объекты использует в качестве входных данных выходные данные другого объекта
5. Смешанное - данные одного объекта используются в другом объекте с неясными целями
6. Патологическое - один объект влияет на внутреннюю реализацию другого или зависит от этой реализации

Комментарий

Low coupling основывается на ООП – так как система состоит из взаимодействующих между собой объектов, это взаимодействие должно быть “адекватно” организовано

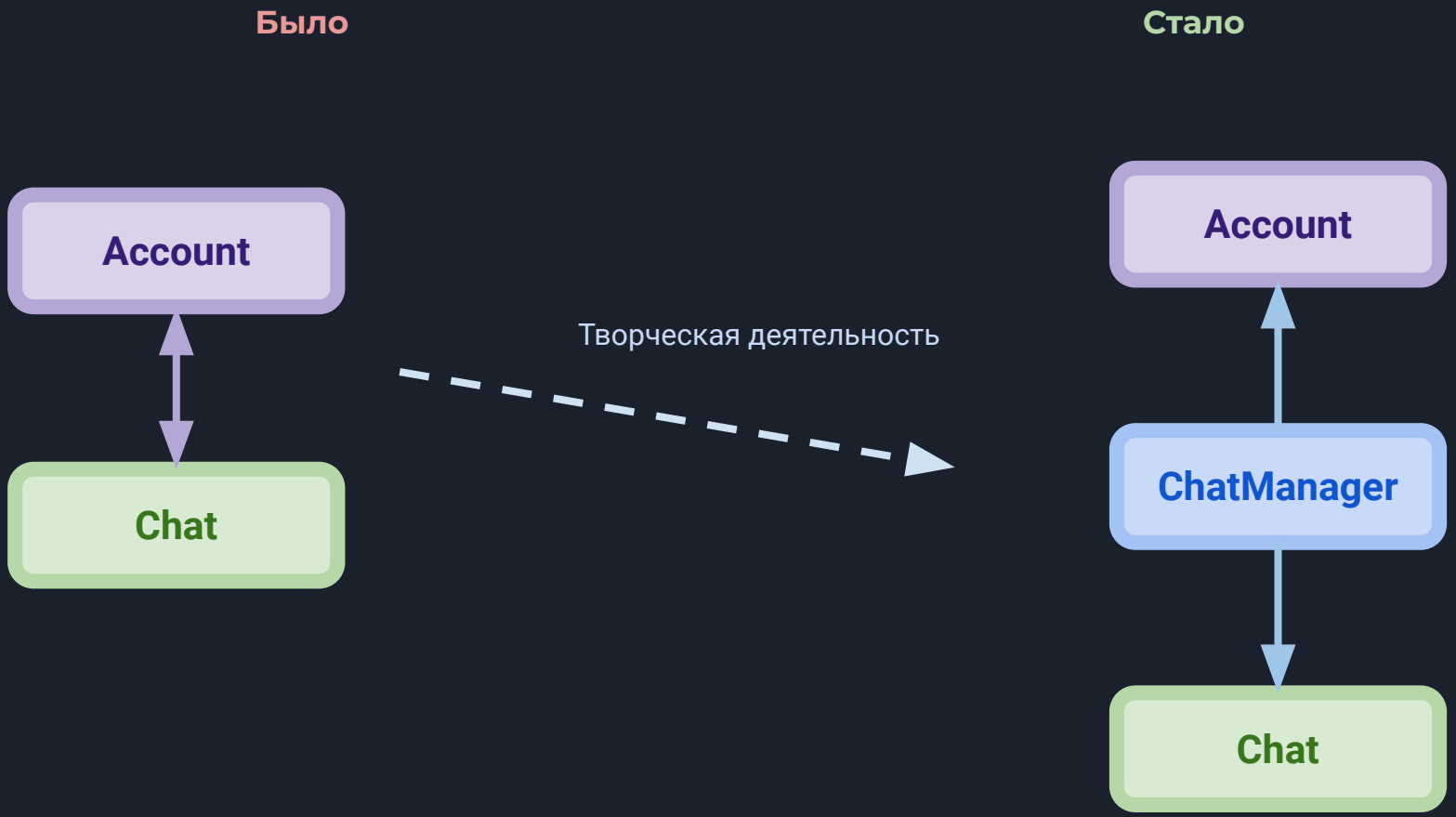
Кажется, зачастую будет полезно определить строгие правила (“протоколы”) взаимодействия между объектами

Так как принцип самый абстрактный из всех, большинство из оставшихся ссылается на него. Получается, что Low coupling заметно выделяется из всей группы принципов

4. Indirection / Перенаправление

Если между классами существует сложная взаимосвязь, для организации взаимодействия между ними лучше использовать класс-посредник

+ Лучше соблюдаем High cohesion и Low coupling


Было
Стало
Account
Chat
Творческая деятельность
Account
ChatManager
Chat

Комментарий

Indirection реализуется посредством вынесения логики взаимодействия объектов в отдельные классы

Благодаря классам-посредникам не всегда снижается общее количество связей, но точно снижается плотность связей

Однако, при переходе в крайность – полном разделении поведения и данных – нарушается Information expert, и значительно повышается риск потери консистентности данных (так как условный DTO не способен следить за корректностью своего состояния)

5. Protected Variations / Устойчивость к изменениям

Если предполагается, что какая-то компонента системы будет часто изменяться, то другие компоненты должны зависеть только от постоянного интерфейса этой компоненты

+ Можно легко заменять компоненты системы, используя различные реализации интерфейсов

Комментарий

Protected variations основывается на принципе абстракции

В книге содержится предостережение о том, что чрезмерное использование абстракции (агрессивный "future-proofing") может быть избыточно. Однако я считаю, что с большинством объектов, отличных от объектов доменной области, то есть с “сервисами” лучше взаимодействовать через интерфейсы, так как есть большая вероятность, что сервисы будут меняться (например, для тестирования могут понадобиться их эмуляторы). Также, dependency inversion стимулирует использование Indirection там, где это действительно нужно и предостерегает от нарушения High cohesion

Похожие принципы: DIP и OCP из SOLID

6. Polymorphism / Полиморфизм

Вариативное поведение, возникающее при использовании объектов различных классов должно быть реализовано с помощью полиморфизма [подтипов]

+ Можно легко изменять поведения системы, используя различные подклассы-реализации ее составляющих

Комментарий

Polymorphism основывается на принципе полиморфизма [ООП] (внезапно)

В книге содержится предостережение о том, что чрезмерное использование абстрактных классов может быть избыточно. Однако я считаю, что создание базового абстрактного класса будет полезно в большинстве из случаев, в которых возможно использование нескольких технологически различных реализаций одного и того же интерфейса, при этом в значительной степени имеющих общее поведение

Как возможные примеры:

1. *Сервисы, которые эмулируются при тестирования (снова)*
2. *Сервисы, использующие различные варианты внешних API*

7. Pure fabrication / Чистая выдумка

Иногда полезно создавать специальные классы, у которых нет реальных аналогов в рассматриваемой доменной области

+ Проявив максимальное творчество, применили Indirection

Комментарий

Pure fabrication основывается на Indirection

Применяется в ситуациях, когда Information expert не помогает

Мне лично кажется избыточным, так как в основном уточняет Indirection

(при этом в книге описывается до Indirection)

8. Creator / Создатель

Объект X может создавать экземпляры класса A только если выполнен какой-то из критериев (а лучше сразу несколько):

1) X содержит или агрегирует объекты класса A
2) X записывают объекты класса A
3) X “плотно” (closely) использует объекты класса A
4) X имеет все необходимые для инициализации объекта данные

\+ Применили information expert по отношению к созданию объектов

Комментарий

Несмотря на подробное описание правил распределения важной ответственности в ООП – ответственности создания объектов – по большей части Creator вытекает из Information expert

Мне лично кажется недостаточно абстрактным по сравнению с остальными

9. Controller / Контроллер

Пользовательские действия должны первоначально обрабатываться объектами специального класса (контроллера), которые делегируют задачи другим объектам с целью соблюдения High cohesion

+ Применили Pure fabrication по отношению к обработке действий пользователя

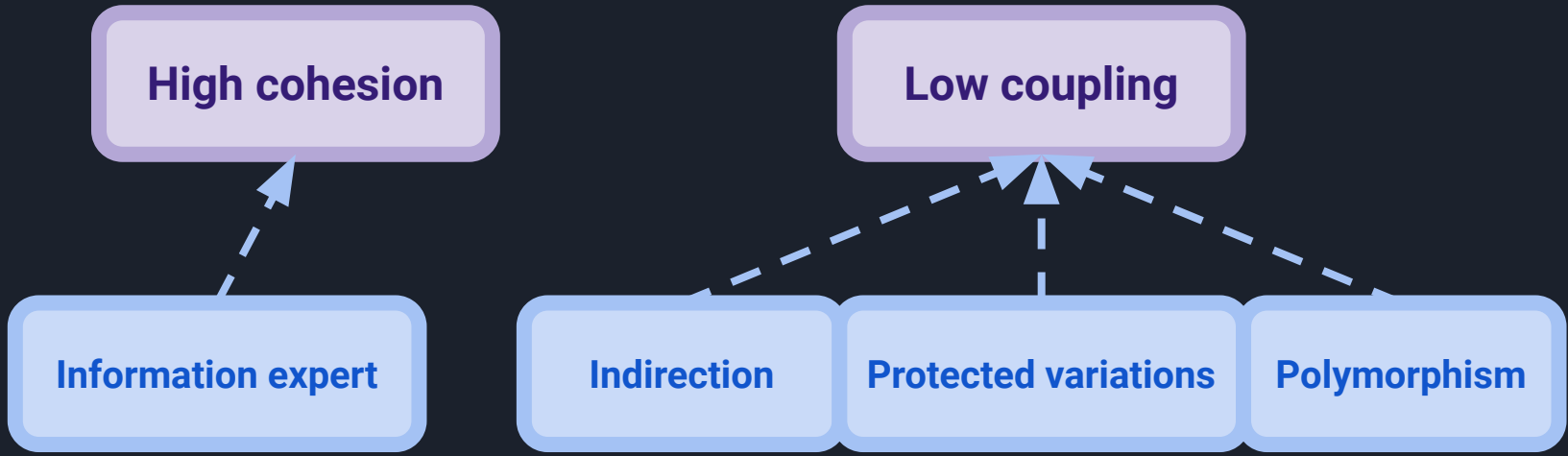
Комментарий

Controller основывается на Pure Fabrication

Мне лично кажется слишком прикладным (хоть и важным), и поэтому резко выделяющимся из всех паттернов

(при этом в книге описывается до High Cohesion)

Как бы я структурировал паттерны



Добро пожаловать в ад
или
Антипаттерны

Искусство войны с ООП

1. Anemic Domain Model - объекты доменной области не имеют поведения, вся бизнес-логика и валидация данных находится на стороне
2. BaseBean - создание класса-утилиты и наследование от него объектов вместо делегирования к нему
3. Call super - при переопределении метода требуется вызывать соответствующий виртуальный метод базового класса
4. Circle-ellipse problem - использование наследования для объекта, в реальности не являющимся полными наследником
5. Circular dependency - создание необязательных зависимостей между объектами
6. Constant interface - использование интерфейсов для хранения констант
7. God object - концентрация слишком большого количества функций в одном классе

Искусство войны с ООП

8. Interface soup - объединение нескольких не связанных друг с другом интерфейсов в один
9. Object cesspool - переиспользование объектов, находящихся в непригодном для этого состоянии
10. Poltergeists - использование объектов, необходимых только для передачи данных другим объектам
11. Sequential coupling - использование методов, которые могут быть вызваны только строго в определенном порядке
12. Singletonitis - неуместное использование Singleton
13. Stub - попытка подогнать существующий объект под плохо подходящий ему интерфейс вместо создания нового
14. Yo-yo problem - чрезмерная фрагментация логически связанного кода

Инструкция по усложнению процесса разработки

1. Copy-paste - копирование кода вместо создания общих решений
2. Golden Hammer - восприятие некоторого инструмента как универсального
3. Improbability factor - предположение того, что известная потенциальная ошибка никогда не возникнет по некоторым причинам
4. Reinventing the wheel - пересоздание уже существующих решений
5. Invented here - избегание разработки инновационных и неочевидных решений, использование только уже существующего
6. Premature optimization - попытки оптимизировать решение на раннем этапе, приводящие к избыточному усложнению системы
7. Programming by permutation - решение проблем путем итеративного внесения в программу небольших изменений с надеждой что проблема пропадет
8. Bug-driven development - чрезмерное фокусирование на правке багов, игнорирование других задач

Используемые источники

1. Larman, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3rd ed., Pearson, 2004. (основной источник)
2. Yourdon, Edward, and Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press, 1978.
3. International Organization for Standardization. (2017). Systems and software engineering — Vocabulary (ISO/IEC/IEEE 24765:2017). **https://www.iso.org/standard/71952.html**
4. **GRASP Wiki**
5. **Антипаттерны**

Inspired by