Задание 3

Отчёт

Беркмонас Иван, Радченко Алексей

   Группа 023 ИСП

Оглавление

[Введение 3](#_Toc216316139)

[1 Понятие паттерна проектирования 4](#_Toc216316140)

[2 Классификации паттернов проектирования 5](#_Toc216316141)

[3 Подробный разбор основных паттернов проектирования 6](#_Toc216316142)

[3.1 Порождающие паттерны 6](#_Toc216316143)

[3.1.1 Фабричный метод 6](#_Toc216316144)

[3.1.2 Абстрактная фабрика 7](#_Toc216316145)

[3.1.3 Одиночка (Singleton) 8](#_Toc216316146)

[3.2 Структурные паттерны 9](#_Toc216316147)

[3.2.1 Адаптер 9](#_Toc216316148)

[3.2.2 Декоратор 10](#_Toc216316149)

[3.2.3 Фасад 10](#_Toc216316150)

[3.3 Поведенческие паттерны 11](#_Toc216316151)

[3.3.1 Стратегия 11](#_Toc216316152)

[3.3.2 Наблюдатель 12](#_Toc216316153)

[3.3.3 Шаблонный метод 12](#_Toc216316154)

[4 Реализация программных модулей 14](#_Toc216316155)

[Программа №1 14](#_Toc216316156)

[Программа №2 21](#_Toc216316157)

[Программа №3 31](#_Toc216316158)

[Заключение 47](#_Toc216316159)

[Список литературы 48](#_Toc216316160)

# Введение

В современной разработке программного обеспечения важную роль играет повторное использование проверённых решений. Одним из инструментов такого повторного использования являются паттерны (шаблоны) проектирования. Они описывают типовые способы построения взаимодействия классов и объектов, которые можно применять в различных проектах, независимо от конкретного языка программирования или предметной области.

Цель данной работы – изучить понятие паттерна проектирования, рассмотреть основные классификации паттернов, а также подробно разобрать примеры порождающих, структурных и поведенческих паттернов на концептуальном уровне.

# 1 Понятие паттерна проектирования

Паттерн проектирования – это типичное, многократно проверенное решение часто встречающейся задачи проектирования программного обеспечения. Паттерн не является готовым фрагментом кода, который можно просто скопировать и вставить в программу. Это, прежде всего, описание структуры классов и их взаимодействия, то есть своего рода «рецепт» построения решения.

Важно отличать паттерн проектирования от алгоритма и библиотеки. Алгоритм описывает последовательность шагов для решения вычислительной задачи. Библиотека предоставляет готовые функции и классы для выполнения конкретных операций. Паттерн же задаёт архитектурный каркас взаимодействия объектов: какие роли есть в системе, какие объекты за что отвечают и как они обмениваются данными.

Использование паттернов позволяет повысить гибкость и расширяемость кода, снизить связность между компонентами, а также улучшить понимание архитектуры за счёт использования общеизвестных терминов. Когда разработчики говорят, что в проекте используется, например, паттерн «Наблюдатель», они кратко описывают целую схему взаимодействия объектов.

# 2 Классификации паттернов проектирования

Классическая и наиболее распространённая классификация паттернов проектирования предложена группой авторов, известной как «банда четырёх» (Gamma, Helm, Johnson, Vlissides). В соответствии с этой классификацией все паттерны делятся на три большие группы в зависимости от своего назначения.

1) Порождающие паттерны – описывают гибкие способы создания объектов, скрывают детали конструирования и позволяют подменять конкретные классы;

2) Структурные паттерны – определяют, как классы и объекты могут комбинироваться в более крупные структуры, упрощают построение сложных иерархий и повторное использование компонентов;

3) Поведенческие паттерны – описывают модели взаимодействия объектов между собой, распределение обязанностей и способы организации обмена сообщениями.

Кроме основной классификации, в литературе можно встретить дополнительные группы, например паттерны архитектурного уровня, конкурентные паттерны, интеграционные паттерны и другие. Однако в рамках данной работы основное внимание уделяется классическим порождающим, структурным и поведенческим паттернам.

# 3 Подробный разбор основных паттернов проектирования

3.1 Порождающие паттерны

Порождающие паттерны отвечают за создание объектов. Они позволяют вынести логику конструирования из клиентского кода, тем самым ослабляя связь между клиентом и конкретными классами. Благодаря этому программу проще изменять и расширять: можно подменять используемые классы, почти не затрагивая остальной код.

3.1.1 Фабричный метод

Назначение: определить интерфейс для создания объектов некоторого типа, позволяя подклассам решать, какой конкретный класс создавать. Фабричный метод переносит ответственность за создание объектов из клиента в специализированный «фабричный» класс.

Основная идея заключается в том, что вместо создания объектов через оператор конструктора в клиентском коде, клиент обращается к методу фабрики. Конкретные подклассы фабрики переопределяют этот метод и возвращают нужные им типы объектов. Клиенту достаточно работать с абстракциями, не зная о том, какой конкретно класс используется в данный момент.

Типичные участники паттерна: абстрактный создатель (определяет фабричный метод), конкретные создатели (переопределяют фабричный метод), а также абстрактный продукт и конкретные продукты. Клиент работает через абстрактный создатель и абстрактный продукт.

Преимущества: фабричный метод позволяет изолировать создание объектов от их использования, упрощает расширение и модификацию системы. Недостаток – возможное увеличение количества классов из-за необходимости создавать отдельные фабрики для разных продуктов.

Применение: фабричный метод полезен тогда, когда системе заранее неизвестно, объекты каких классов ей придётся создавать, когда требуется обеспечить расширяемость ряда продуктов или скрыть детали их создания от клиента.

3.1.2 Абстрактная фабрика

Назначение: предоставить интерфейс для создания семейств взаимосвязанных объектов без указания их конкретных классов. В отличие от простого фабричного метода, абстрактная фабрика сразу отвечает за выпуск целых семейств продуктов, которые должны использоваться совместно.

Основная идея состоит в том, что клиент работает с фабрикой, которая умеет создавать согласованный набор объектов: например, различные элементы графического интерфейса для конкретной операционной системы. Смена фабрики позволяет подменить сразу всё семейство продуктов, сохранив единый стиль и совместимость компонентов.

Типичные участники: интерфейс абстрактной фабрики, конкретные фабрики для каждого семейства продуктов, абстрактные и конкретные продукты, а также клиент, который использует только абстракции, не завися от реализаций.

Преимущества: абстрактная фабрика поддерживает принцип инверсии зависимостей, облегчает смену семейств продуктов, гарантирует совместимость созданных объектов. Недостаток – сложность расширения, если нужно добавить новый тип продукта во все семейства.

Применение: абстрактная фабрика подходит для кроссплатформенных приложений, для систем с несколькими режимами работы (например, «светлая» и «тёмная» тема оформления), а также в случаях, когда необходимо строгое соответствие между создаваемыми объектами.

3.1.3 Одиночка (Singleton)

Назначение: гарантировать, что у класса существует только один экземпляр, и предоставить глобальную точку доступа к этому экземпляру. Паттерн «Одиночка» часто используется для объектов, которые должны быть уникальными в рамках всего приложения.

Основная идея заключается в скрытии конструктора класса и предоставлении статического метода, который возвращает единственный экземпляр. При первом обращении объект создаётся, а при последующих – возвращается уже существующий экземпляр.

Преимущества: контроль количества экземпляров, удобная точка доступа, возможность ленивой инициализации. Недостатки: потенциальное нарушение принципа единой ответственности, усложнение тестирования и скрытая глобальная зависимость.

Применение: одиночка может использоваться для объектов конфигурации, логгеров, пулов соединений, но применять его следует осторожно, чтобы не превращать весь код в набор неявных глобальных зависимостей.

3.2 Структурные паттерны

Структурные паттерны описывают способы построения связей между классами и объектами. Они помогают упростить архитектуру, скрыть сложность реализации за простым интерфейсом и облегчить повторное использование компонентов.

3.2.1 Адаптер

Назначение: преобразовать интерфейс одного класса в другой интерфейс, ожидаемый клиентами. Благодаря паттерну «Адаптер» классы с несовместимыми интерфейсами могут работать вместе.

Основная идея состоит в создании специального класса-обёртки, который реализует требуемый клиентом интерфейс и внутри делегирует вызовы объекту с несовместимым интерфейсом. Таким образом, клиент продолжает работать с удобным для него контрактом, а адаптер занимается переводом вызовов.

Преимущества: позволяет повторно использовать существующий код, не изменяя его, упрощает интеграцию сторонних библиотек. Недостаток – возможное появление большого количества адаптеров при множестве различных интерфейсов.

Применение: адаптер широко используется при интеграции с чужими библиотеками, при миграции со старых API на новые, а также при необходимости временно связать несовместимые компоненты.

3.2.2 Декоратор

Назначение: динамически добавлять объекту новые обязанности, не изменяя его класс. Паттерн «Декоратор» позволяет гибко комбинировать дополнительные функции, оборачивая исходный объект в цепочку декораторов.

Основная идея заключается в том, что декоратор реализует тот же интерфейс, что и основной объект, и содержит ссылку на этот объект. При вызове методов декоратор может выполнять дополнительную логику до или после делегирования вызова исходному объекту.

Преимущества: декоратор предоставляет альтернативу множественному наследованию и взрывному росту количества подклассов, так как новые поведения можно добавлять путём комбинирования декораторов. Недостатки – усложнение структуры и необходимость понимать порядок обёртывания объектов.

Применение: декораторы полезны при добавлении функциональности, связанной с логированием, кешированием, проверкой прав доступа, форматированием данных и другими поперечными задачами, которые не хочется жёстко «вшивать» в основной класс.

3.2.3 Фасад

Назначение: предоставить унифицированный, упрощённый интерфейс к сложной подсистеме. Фасад скрывает множество деталей реализации и зависимости между компонентами, предлагая клиенту один удобный входной пункт.

Основная идея состоит в создании отдельного класса-фасада, который инкапсулирует работу с набором сложных объектов. Клиент вызывает методы фасада, а тот, в свою очередь, координирует взаимодействие внутренних компонентов.

Преимущества: уменьшение связности между клиентским кодом и сложной подсистемой, упрощение использования API, улучшение читаемости и поддержки кода. Недостаток – риск превратить фасад в «большой объект», делающий слишком много.

Применение: фасад используют для инкапсуляции сложных библиотек, подсистем доступа к данным, модулей работы с сетью и других частей системы, где клиенту достаточно нескольких высокоуровневых операций.

**3.3 Поведенческие паттерны**

Поведенческие паттерны описывают способы организации взаимодействия объектов, распределение обязанностей и механизмы обмена информацией между ними. Цель таких паттернов – сделать это взаимодействие гибким и легко изменяемым.

3.3.1 Стратегия

Назначение: определить семейство алгоритмов, инкапсулировать каждый из них и обеспечить их взаимозаменяемость. Паттерн «Стратегия» позволяет изменять алгоритм независимо от клиентов, которые его используют.

Основная идея состоит в том, что набор взаимозаменяемых алгоритмов реализуется в виде отдельных классов, которые имеют общий интерфейс. Клиентский объект хранит ссылку на выбранную стратегию и делегирует ей выполнение работы. При необходимости стратегию можно заменить на другую во время выполнения программы.

Преимущества: стратегия избавляет от громоздких условных операторов, упрощает добавление новых алгоритмов и поддерживает принцип открытости/закрытости. Недостаток – увеличение числа классов и необходимость выбора подходящей стратегии.

Применение: паттерн часто используется для реализации различных вариантов сортировки, способов расчёта стоимости, политик кеширования, способов сериализации данных и других взаимозаменяемых алгоритмов.

3.3.2 Наблюдатель

Назначение: организовать зависимость «один-ко-многим» между объектами таким образом, чтобы при изменении состояния одного объекта все зависящие от него объекты автоматически уведомлялись и обновлялись. Паттерн «Наблюдатель» реализует механизм подписки и публикации уведомлений.

Основная идея состоит в наличии субъекта (издателя), который хранит список наблюдателей (подписчиков). Когда состояние субъекта меняется, он оповещает всех подписчиков через заранее определённый интерфейс. Наблюдатели, в свою очередь, реагируют на полученное уведомление соответствующим образом.

Преимущества: слабая связность между источником событий и их обработчиками, возможность динамически подписывать и отписывать наблюдателей, расширяемость системы. Недостатки – сложность отладки при большом количестве подписчиков и опасность циклических уведомлений.

Применение: наблюдатель активно используется в графических пользовательских интерфейсах, системах событий, механизмах подписки на изменения данных, а также во многих фреймворках и библиотечных реализациях реактивного программирования.

3.3.3 Шаблонный метод

Назначение: определить скелет алгоритма в базовом классе, перекладывая реализацию некоторых его шагов на подклассы. Паттерн «Шаблонный метод» позволяет переопределять отдельные шаги алгоритма, не меняя его общей структуры.

Основная идея заключается в том, что базовый класс содержит общий алгоритм, реализованный в виде последовательности вызовов шагов. Часть шагов реализована в базовом классе, а часть объявлена абстрактной и должна быть реализована в подклассах. Таким образом, разные подклассы могут по-своему реализовывать отдельные шаги, сохраняя общий порядок выполнения.

Преимущества: повторное использование общей части алгоритма, центризация базовой логики в одном месте, снижение дублирования кода. Недостатки – жёсткая связь между базовым классом и подклассами, усложнение иерархий наследования.

Применение: шаблонный метод применяют для описания стандартных процессов, которые могут иметь вариации в отдельных шагах, например, обработки файлов, проведения транзакций, последовательностей операций в бизнес‑процессах и так далее.

# 4 Реализация программных модулей

Программа №1

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.Рисунок 1

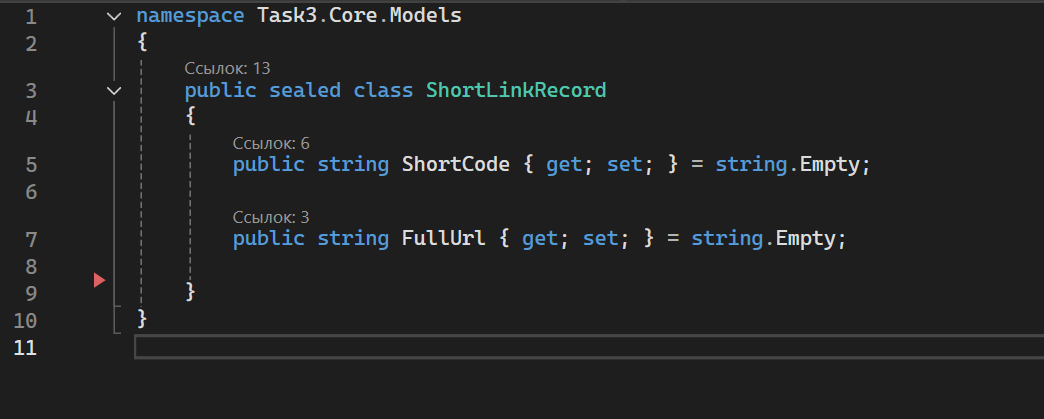


Рисунок 2

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3

Изображение выглядит как текст, снимок экрана, меню

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 5

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 6

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 7

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 8

Программа №2

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 9

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 10

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 11

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 12

Изображение выглядит как текст, снимок экрана, меню

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 13

Изображение выглядит как текст, снимок экрана, меню

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 14

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 15

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 16

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 17

Изображение выглядит как текст, снимок экрана, меню

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 18

Программа №3

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 19

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 20

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 21

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 22

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 23

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 24

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 25

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 26

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 27

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 28

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 29

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 30

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 31

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 32

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 33

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 34

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 35

# Заключение

В ходе теоретической части работы были рассмотрены основные понятия, связанные с паттернами проектирования, изучена их классическая классификация на порождающие, структурные и поведенческие паттерны. Подробно разобраны примеры наиболее часто используемых паттернов в каждой из групп, описаны их назначение, ключевые идеи, участники, преимущества, недостатки и типичные области применения.

Применение паттернов проектирования позволяет создавать более гибкие, расширяемые и сопровождаемые программные системы. Понимание паттернов упрощает коммуникацию между разработчиками, так как сложные архитектурные решения можно описывать краткими, общеизвестными терминами. Освоение паттернов является важной частью профессионального роста разработчика и основой для изучения более сложных архитектурных подходов.

# Список литературы

1) [Каталог паттернов проектирования](https://refactoring.guru/ru/design-patterns)

2) [Шпаргалка по шаблонам проектирования / Хабр](https://habr.com/ru/articles/210288/)

3) [Паттерны проектирования: какие бывают и как выбрать нужный](https://gb.ru/blog/patterny-proektirovaniya/)

P.S.

Стоит отметить, что первого источника более чем хватает. Не вижу смысла дублировать ссылку на этот проект с разными страницами под каждые классы паттернов. Остальные ссылки, по большей части, существуют для вида.