Министерство Образования, Культуры и Исследований Республики Молдова

Технический Университет Молдовы

Факультет Вычислительной Техники, Информатики и Микроэлектроники

Департамент Программной Инженерии и Автоматики

**Курсовая работа**

по предмету «Техника и Методы Проектирования Систем»

**Тема:**

«Проектирование и реализация паттернов в компьютерной игре, с элементами RPG, на движке Unity»

"Design and implementation of patterns in a computer game, with RPG elements, on the Unity engine"

Выполнил: ст. гр. TI-197 Шарафудинов Николай

Проверил: унив. ассист. Булдумак Олег

Кишинев 2022

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc102937433)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 3](#_Toc102937434)

[1.1 Важность темы 3](#_Toc102937435)

[1.2 Облачная инфраструктура / платформа / сервис, как услуга 4](#_Toc102937436)

[1.3 Цели создания системы 6](#_Toc102937437)

[1.4 Требования к системе 7](#_Toc102937438)

[2 МОДЕЛИРОВАНИЕ СИСТЕМЫ 10](#_Toc102937439)

[2.1 Функциональное назначение системы 10](#_Toc102937440)

[2.2 Взаимодействие элементов системы 10](#_Toc102937441)

[2.3 Моделирование физического представления системы 15](#_Toc102937442)

[3 РЕАЛИЗАЦИЯ СИСТЕМЫ 17](#_Toc102937443)

[3.1 Структура проекта 17](#_Toc102937444)

[3.2 Описание классов системы 17](#_Toc102937445)

[ЗАКЛЮЧЕНИЕ 21](#_Toc102937446)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22](#_Toc102937447)

[ПРИЛОЖЕНИЕ А 24](#_Toc102937448)

[Скрипт 1 24](#_Toc102937449)

[Скрипт 2 25](#_Toc102937450)

[Скрипт 3 25](#_Toc102937451)

# ВВЕДЕНИЕ

Проектирование объектно-ориентированных программ - нелегкое дело, а если их нужно использовать повторно, то все становится еще сложнее. Необходимо подобрать подходящие объекты, отнести их к различным классам, соблюдая разумную степень детализации, определить интерфейсы классов и иерархию наследования и установить существенные отношения между классами. Дизайн должен, с одной стороны, соответствовать решаемой задаче, с другой - быть общим, что-бы удалось учесть все требования, которые могут возникнуть в будущем. Хотелось бы также избежать вовсе или, по крайней мере, свести к минимуму необходимость перепроектирования.

Что представляют собой паттерны проектирования? Паттерн представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования. В данном случае предполагается, что есть некоторый набор общих формализованных проблем, которые довольно часто встречаются, и паттерны предоставляют ряд принципов для решения этих проблем. Что же дает нам применение паттернов? При написании программ мы можем формализовать проблему в виде классов и объектов и связей между ними. И применить один из существующих паттернов для ее решения. В итоге нам не надо ничего придумывать. У нас уже есть готовый шаблон, и нам только надо его применить в конкретной программе. Хорошая программа предполагает использование паттернов. Однако не всегда паттерны упрощают и улучшают программу. Неоправданное их использование может привести к усложнению программного кода, уменьшению его качества. Паттерн должен быть оправданным и эффективным способом решения проблемы.

Существует множество различных паттернов, которые решают разные проблемы и выполняют различные задачи. Но по своему действию их можно объединить в ряд групп. Рассмотрим некоторые группы паттернов. В основу классификации основных паттернов положена цель или задачи, которые определенный паттерн выполняет.

Хороший шаблон проектирования должен быть таким, чтобы его можно было использовать с большинством языков (если не со всеми) в зависимости от характеристик языка. Чрезвычайно важно то, что любой шаблон проектирования необходимо использовать очень осторожно — если он применён в ненадлежащем месте, то его действие может быть разрушительным и породить много проблем для вас. Однако применённый в нужном месте в нужное время он может стать вашим спасителем.

# ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

В данной главе будет рассказано про важность темы, а также про классификацию паттернов, а также про каждый из видов отдельно.

## 1.1 Важность темы

По словам Кристофера Александра, «Каждый паттерн описывает проблему, которая встречается снова и снова в нашем окружении, и затем описывает суть решения этой проблемы, так, что вы можете использовать это решение миллион раз, никогда не делая это дважды одинаковым образом.».[12] Хотя Александр имел в виду паттерны, возникающие при проектировании зданий и городов, но его слова верны и в отношении паттернов объектно-ориентированного проектирования.

Шаблоны проектирования — это один из инструментов разработчика, который помогает ему сэкономить время и сделать более качественное решение. Как и любой другой инструмент, в одних руках он может принести много пользы, а в других — один только вред.

Можно вполне успешно работать, не зная ни одного паттерна. Более того, могли уже не раз реализовать какой-то из паттернов, даже не подозревая об этом.

Но осознанное владение инструментом как раз и отличает профессионала от любителя. Можете забить гвоздь молотком, а можете и дрелью, если сильно постараетесь. Но профессионал знает, что главная фишка дрели совсем не в этом. Итак, зачем же знать паттерны?

* Проверенные решения. Вы тратите меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений вы смогли бы додуматься и сами, но многие могут быть для вас открытием.
* Стандартизация кода. Вы делаете меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.
* Общий программистский словарь. Вы произносите название паттерна, вместо того чтобы час объяснять другим программистам, какой крутой дизайн вы придумали и какие классы для этого нужны.[4]

## **1.2 Классификация паттернов проектирования**

Прежде всего при решении какой-нибудь проблемы надо выделить все используемые сущности и связи между ними и абстрагировать их от конкретной ситуации. Затем надо посмотреть, вписывается ли абстрактная форма решения задачи в определенный паттерн. Например, суть решаемой задачи может состоять в создании новых объектов. В этом случае, возможно, стоит посмотреть на порождающие паттерны. Причем лучше не сразу взять какой-то определенный паттерн - первый, который показался нужным, а посмотреть на несколько родственных паттернов из одной группы, которые решают одну и ту же задачу.

При этом важно понимать смысл и назначение паттерна, явно представлять его абстрактную организацию и его возможные конкретные реализации. Один паттерн может иметь различные реализации, и чем чаще вы будете сталкиваться с этими реализациями, тем лучше вы будете понимать смысл паттерна. Но не стоит использовать паттерн, если вы его не понимаете, даже если он на первый взгляд поможет вам в решении задачи.

И в конечном счете надо придерживаться принципа KISS (Keep It Simple, Stupid) - сохранять код программы по возможности простым и ясным. Ведь смысл паттернов не в усложнении кода программы, а наоборот в его упрощении. [2]

## 1.3 Порождающие паттерны проектирования

Порождающие шаблоны — шаблоны проектирования, которые имеют дело с процессом создания объектов. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять наследуемый класс, а шаблон, порождающий объекты, делегирует экземпляр класса другому объекту. Всего порождающих паттернов шесть:

* Абстрактная фабрика (Abstract Factory) — это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.
* Строитель (Builder) — это порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.
* Фабричный метод (Factory Method) — это порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.
* Бассейн Объектов (Object Pool) – Объединение объектов в пул может обеспечить значительный прирост производительности; он наиболее эффективен в ситуациях, когда стоимость инициализации экземпляра класса высока, скорость создания экземпляра класса высока, а количество экземпляров, используемых в любой момент времени, невелико.
* Прототип (Prototype) — это порождающий паттерн проектирования, который позволяет копировать объекты, не вдаваясь в подробности их реализации.
* Одиночка (Singleton) — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

При выполнение своей курсовой работы были использованы такие порождающие паттерны как: Строитель и Фабрика.

## 1.4 Структурные паттерны проектирования

Структурные шаблоны в основном связаны с композицией объектов, другими словами, с тем, как сущности могут использовать друг друга. Ещё одним объяснением было бы то, что они помогают ответить на вопрос «Как создать программный компонент?».

Структурные шаблоны — шаблоны проектирования, в которых рассматривается вопрос о том, как из классов и объектов образуются более крупные структуры. Всего структурных паттернов семь:

* Адаптер (Adapter) – Преобразование интерфейса класса в другой интерфейс, ожидаемый клиентами. Адаптер позволяет классам работать вместе, что в противном случае было бы невозможно из-за несовместимых интерфейсов. Оберните существующий класс новым интерфейсом. Сопротивление соответствует старому компоненту новой системе.
* Мост (Bridge) – Отделите абстракцию от ее реализации, чтобы они могли меняться независимо друг от друга. Опубликуйте интерфейс в иерархии наследования и похороните реализацию в собственной иерархии наследования. Помимо инкапсуляции, к изоляции.
* Компоновщик (Composite) – Объединяйте объекты в древовидные структуры, чтобы представлять иерархии целых частей. Composite позволяет клиентам единообразно обрабатывать отдельные объекты и композиции объектов. Рекурсивная композиция «Каталоги содержат записи, каждая из которых может быть директорией». 1-ко-многим "имеет" вверх по иерархии.
* Декоратор (Decorator) – Динамически прикрепляйте дополнительные обязанности к объекту. Декораторы предоставляют гибкую альтернативу подклассам для расширения функциональности. Определяемое клиентом украшение основного объекта путем его рекурсивной упаковки. Упаковать подарок, положить его в коробку и упаковать коробку.
* Фасад (Facade) – Обеспечить унифицированный интерфейс для набора интерфейсов в подсистеме. Фасад определяет высокоуровневый интерфейс, упрощающий использование подсистемы. Оберните сложную подсистему более простым интерфейсом.
* Приспособленец (Flyweight) – Используйте совместное использование для эффективной поддержки большого количества мелких объектов. Стратегия графического интерфейса Motif по замене тяжелых виджетов легкими гаджетами.
* Заместитель (Proxy) – Предоставьте суррогат или заполнитель для другого объекта, чтобы управлять доступом к нему. Используйте дополнительный уровень косвенности для поддержки распределенного, контролируемого или интеллектуального доступа. Добавьте оболочку и делегирование, чтобы защитить реальный компонент от чрезмерной сложности.

В работе, были реализованы такие структурные паттерны проектирования, как: Декоратор и Мост.

## 1.5 Поведенческие паттерны проектирования

Поведенческие шаблоны связаны с распределением обязанностей между объектами. Их отличие от структурных шаблонов заключается в том, что они не просто описывают структуру, но также описывают шаблоны для передачи сообщений / связи между ними. Или, другими словами, они помогают ответить на вопрос «Как запустить поведение в программном компоненте?»

Поведенческие шаблоны — шаблоны проектирования, определяющие алгоритмы и способы реализации взаимодействия различных объектов и классов. Всего поведенческих паттернов десять:

* Цепочка обязанностей (Chain of Responsibility) – поведенческий шаблон проектирования, предназначенный для организации в системе уровней ответственности.
* Команда (Command) – поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.
* Итератор (Iterator) – поведенческий шаблон проектирования. Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из агрегированных объектов.
* Посредник (Mediator) – поведенческий шаблон проектирования, обеспечивающий взаимодействие множества объектов, формируя при этом слабую связанность, и избавляя объекты, от необходимости явно ссылаться друг на друга.
* Хранитель/Снимок (Memento) – поведенческий шаблон проектирования, позволяющий, не нарушая инкапсуляцию, зафиксировать и сохранить внутреннее состояние объекта так, чтобы позднее восстановить его в этом состоянии.
* Наблюдатель (Observer) – поведенческий шаблон проектирования, также известен как «подчинённые» (Dependents). Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.
* Посетитель (Visitor) – поведенческий шаблон проектирования, описывающий операцию, которая выполняется над объектами других классов. При изменении visitor нет необходимости изменять обслуживаемые классы.
* Стратегия (Strategy) – поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путём определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.
* Состояние (State) – поведенческий шаблон проектирования. Используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния.
* Шаблонный метод (Template Method) – поведенческий шаблон проектирования, определяющий основу алгоритма и позволяющий наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.

В своей работе, из поведенческих паттернов проектирования были выбраны такие паттерны как: Стратегия и Хранитель.

# МОДЕЛИРОВАНИЕ СИСТЕМЫ

В данной главе будет рассмотрено моделирование системы, как и почему были выбраны те или иные паттерны для выполнения данного проекта.

## 2.1 Выбор порождающих паттернов

Из порождающих паттернов проектирования, как было ранее указано, выбор пал на Фабричный метод и Строителя.

Фабричный метод использовался для системы порталов соединяющая уровни Фабричный метод отделяет код производства продуктов от остального кода, который эти продукты использует. Благодаря этому, код производства можно расширять, не трогая основной. Так, чтобы добавить поддержку нового продукта, вам нужно создать новый подкласс и определить в нём фабричный метод, возвращая оттуда экземпляр нового продукта. Преимуществом в использовании Фабричного метода является:

* Избавляет класс от привязки к конкретным классам продуктов.
* Выделяет код производства продуктов в одно место, упрощая поддержку кода.
* Упрощает добавление новых продуктов в программу.
* Реализует принцип открытости/закрытости.

Строитель, в проекте используется для сборки таких объектов как «враг». В игре есть несколько типов врагов, со своими параметрами и со своими изображениями. Паттерн Строитель позволяет собирать объекты пошагово, вызывая только те шаги, которые вам нужны. А значит, больше не нужно пытаться «запихнуть» в конструктор все возможные опции продукта. Строитель можно применить, если создание нескольких представлений объекта состоит из одинаковых этапов, которые отличаются в деталях.

Интерфейс строителей определит все возможные этапы конструирования. Каждому представлению будет соответствовать собственный класс-строитель. А порядок этапов строительства будет задавать класс-директор. Преимуществом в использовании Строителя является:

* Позволяет создавать продукты пошагово.
* Позволяет использовать один и тот же код для создания различных продуктов.
* Изолирует сложный код сборки продукта от его основной бизнес-логики.

## 2.2 Выбор структурных паттернов

В качестве структурных паттернов были выбраны такие паттерны как Мост и Декоратор, как и было сказано ранее в описании структурных паттернов проектирования.

Декоратор используется для наслаивания нового внешнего вида для главного героя в зависимости от его уровня. Объект помещают в обёртки, имеющие дополнительные поведения. Обёртки и сам объект имеют одинаковый интерфейс, поэтому клиентам без разницы, с чем работать — с обычным объектом данных или с обёрнутым. Для обертывания была реализована система, зависящая от уровня персонажа также помимо увеличения здоровья, от новой обертки персонажа, так же растёт и его урон. Преимуществом в использовании Декоратора является:

* Большая гибкость, чем у наследования.
* Позволяет добавлять обязанности на лету.
* Можно добавлять несколько новых обязанностей сразу.
* Позволяет иметь несколько мелких объектов вместо одного объекта на все случаи жизни.

Мост используется для соединения композиции Game manager и Weapon. Мост позволяет разделить монолитный класс на несколько отдельных иерархий. После этого вы можете менять их код независимо друг от друга. Это упрощает работу над кодом и уменьшает вероятность внесения ошибок. Мост предлагает выделить одну из таких плоскостей в отдельную иерархию классов, храня ссылку на один из её объектов в первоначальном классе. В случае выполнения работы есть глобальный класс, в котором прописаны все переменные, которые так или иначе необходимы для логики приложения, такие как опыт персонажа, количество золота, а также какое оружие уровень оружия и т.д. Преимуществом в использовании Моста является:

* Позволяет строить платформа-независимые программы.
* Скрывает лишние или опасные детали реализации от клиентского кода.
* Реализует принцип открытости/закрытости.

## 2.3 Выбор поведенческих паттернов

В качестве поведенческих паттернов проектирования были выбраны такие паттерны как Хранитель и Стратегия, как и было сказано ранее в описании поведенческих паттернов проектирования.

Когда нужно использовать разные вариации какого-то алгоритма внутри одного объекта используется паттерн Стратегия. Именно для этого и был использован данный паттерн ответственный за стратегию ведения боя персонажем. Стратегия позволяет варьировать поведение объекта во время выполнения программы, подставляя в него различные объекты-поведения.

Преимущества поведенческого паттерна проектирования Стратегия:

* Горячая замена алгоритмов на лету.
* Изолирует код и данные алгоритмов от остальных классов.
* Уход от наследования к делегированию.
* Реализует принцип открытости/закрытости.

Хранитель, тот паттерн без которого не обходится ни одна из игр. Возможность сохранить состояние прогресса, а также возможность откатить все к предыдущему состоянию является неотъемлемой частью игрового процесса большинства игр.

Паттерн Снимок позволяет создавать любое количество снимков объекта и хранить их, независимо от объекта, с которого делают снимок. Снимки часто используют не только для реализации операции отмены, но и для транзакций, когда состояние объекта нужно «откатить», если операция не удалась. Паттерн предлагает изготовить снимок самому исходному объекту, поскольку ему доступны все поля, даже приватные.

Преимущества поведенческого паттерна проектирования Снимок/Хранитель:

* Не нарушает инкапсуляции исходного объекта.
* Упрощает структуру исходного объекта. Ему не нужно хранить историю версий своего состояния.

# РЕАЛИЗАЦИЯ СИСТЕМЫ

## 3.1 Описание диаграмм классов

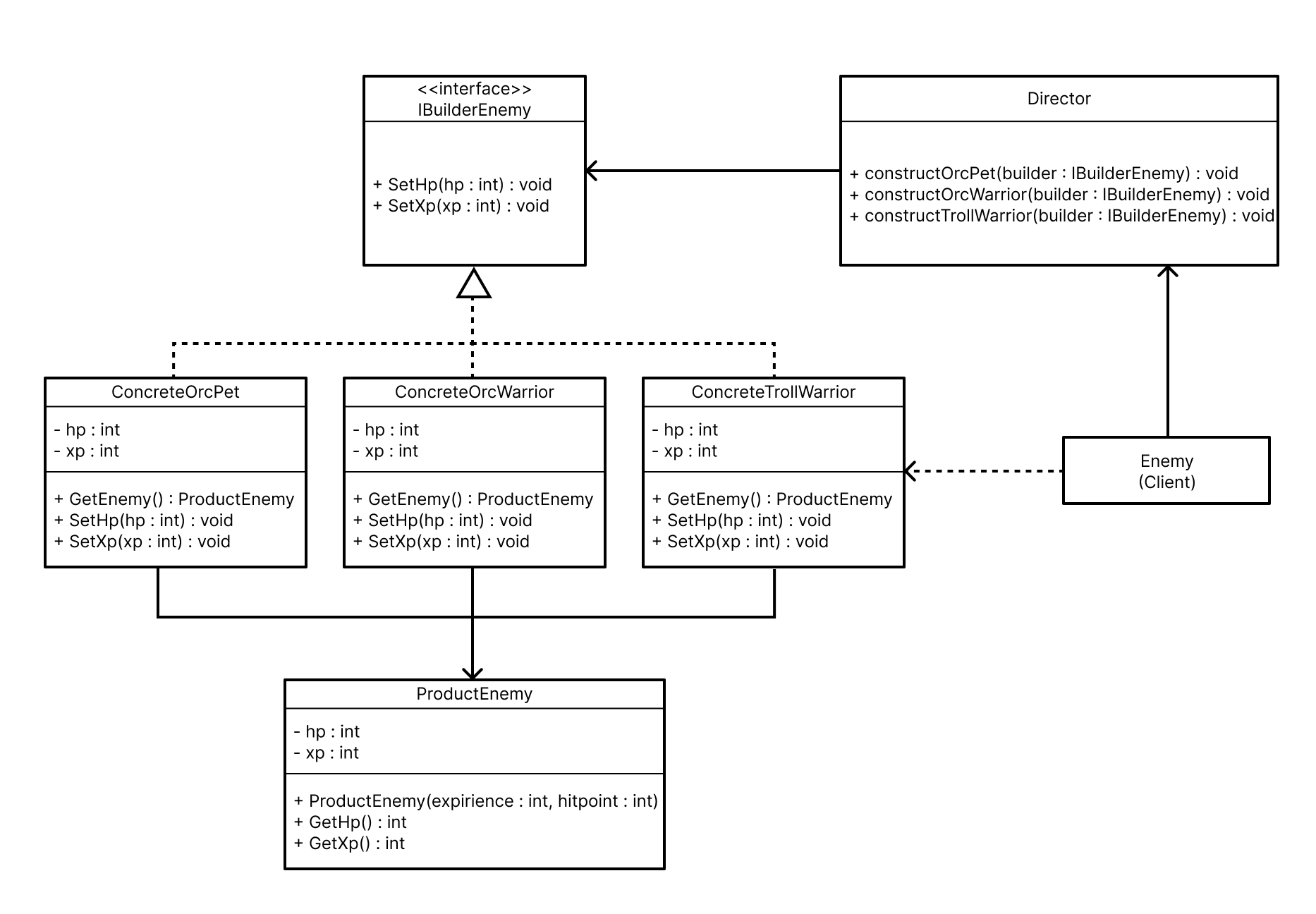


Рисунок 1 - Схема паттерна Строитель

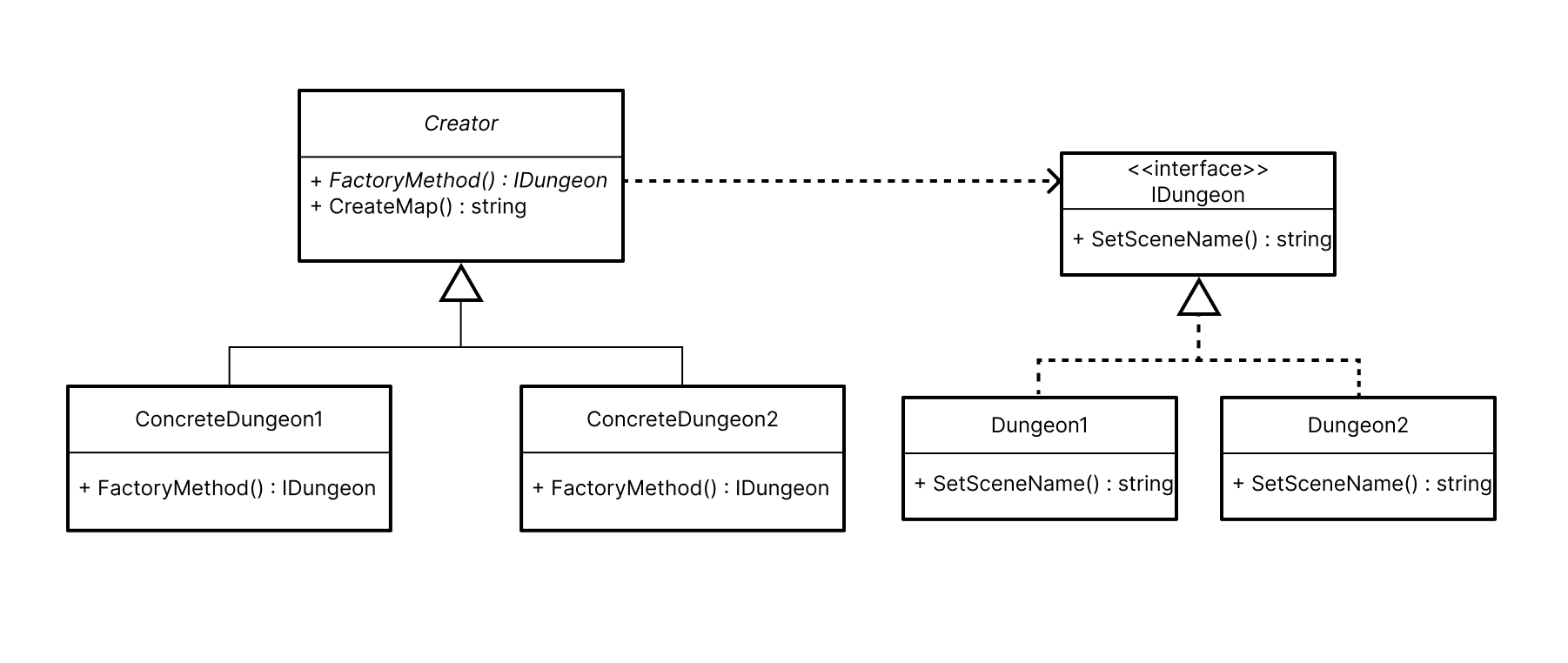


Рисунок 2 - Схема паттерна Фабрика

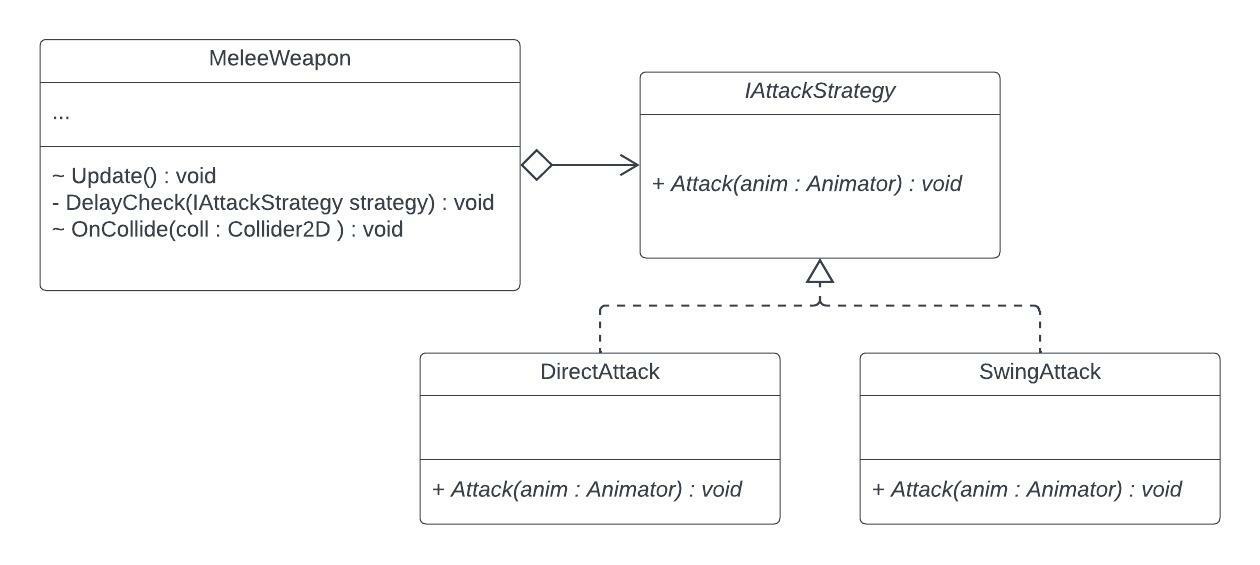


Рисунок 3 - Схема паттерна Стратегия

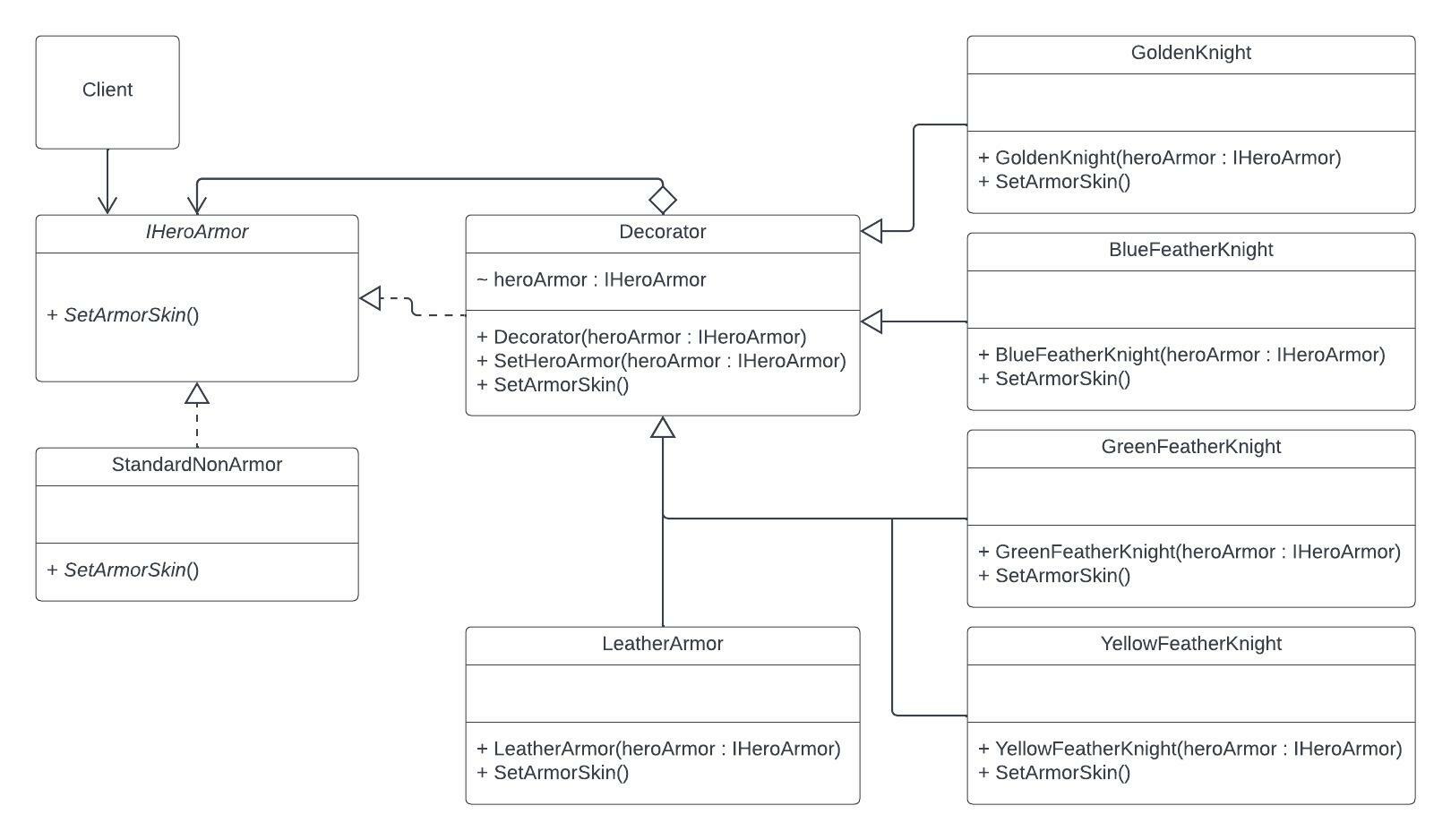


Рисунок 4 - Схема паттерна Декоратор

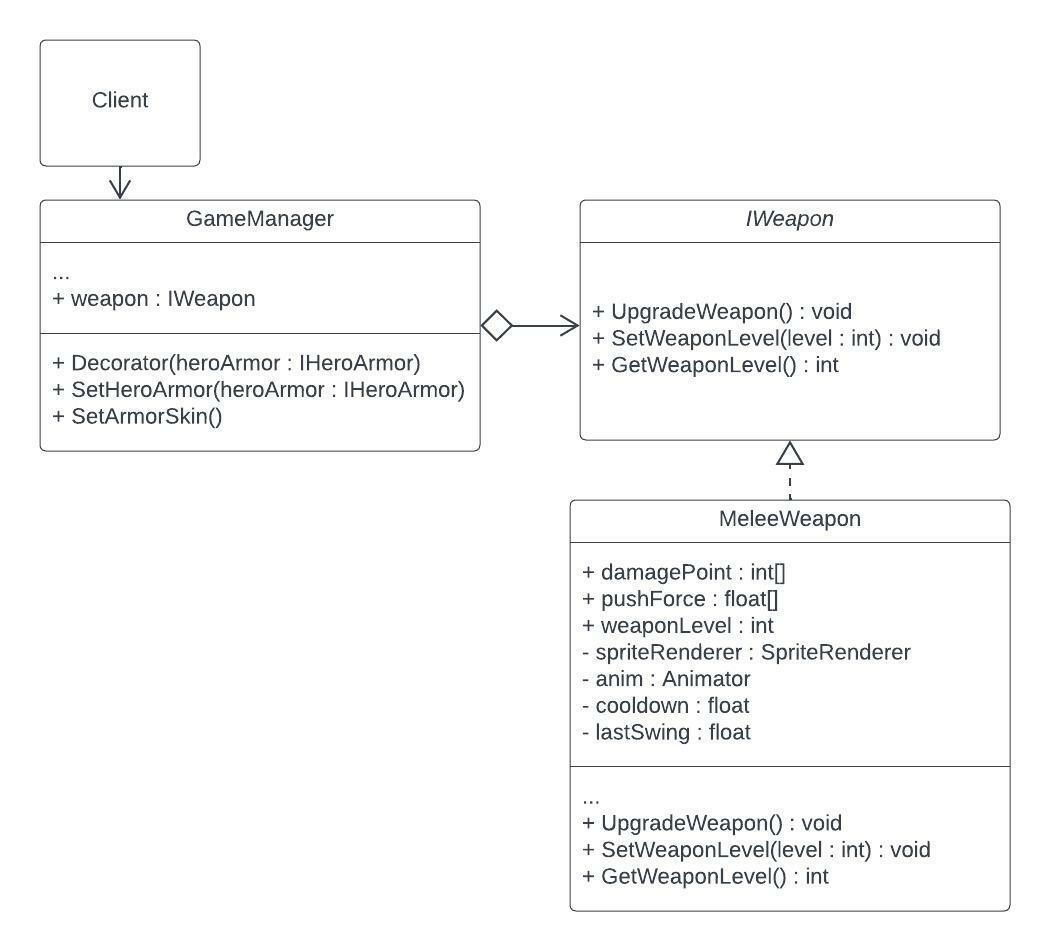


Рисунок 5 - Схема паттерна Мост

## 3.2 Описание классов системы

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое шаблоны проектирования? [Электронный ресурс]. - Режим доступа: <https://habr.com/ru/post/307942/>
2. Введение в паттерны проектирования [Электронный ресурс]. - Режим доступа: <https://metanit.com/sharp/patterns/1.1.php>
3. Шаблоны проектирования: 5 самых известных [Электронный ресурс]. - Режим доступа: <https://apptractor.ru/info/articles/shablony-proektirovanija-5-samyh-izvestnyh.html>
4. Зачем знать паттерны? [Электронный ресурс]. - Режим доступа: <https://refactoring.guru/ru/design-patterns/why-learn-patterns>
5. Элементарные шаблоны проектирования [Электронный ресурс]. - Режим доступа: <http://www.williamspublishing.com/Books/978-5-8459-1818-5.html>
6. Шаблоны проектирования простым языком. Часть первая. Порождающие шаблоны [Электронный ресурс]. - Режим доступа: <https://tproger.ru/translations/design-patterns-simple-words-1/>
7. Шаблоны проектирования простым языком. Часть вторая. Структурные шаблоны [Электронный ресурс]. - Режим доступа: <https://tproger.ru/translations/design-patterns-simple-words-2/>
8. Шаблоны проектирования простым языком. Часть третья. Поведенческие шаблоны [Электронный ресурс]. - Режим доступа: <https://tproger.ru/translations/design-patterns-simple-words-3/>
9. Создание UML диаграмм [Электронный ресурс]. - Режим доступа: <https://lucid.app/documents#/documents?folder_id=home>
10. Шаблоны проектирования "банды четырёх (GoF)" [Электронный ресурс]. - Режим доступа: <https://bool.dev/blog/detail/gof-design-patterns>
11. Design Patterns Elements of Reusable Object-Oriented Software [Электронный ресурс]. - Режим доступа: <https://docs.google.com/file/d/0B6GuCegBf3X3Tm1rZl9BUTduQm8/edit?resourcekey=0-ME3Ni9D9Wae8zLuAbNPx6w>
12. Цитаты известных личностей [Электронный ресурс]. - Режим доступа: <https://ru.citaty.net/tsitaty/2048363-kristofer-aleksander-kazhdyi-pattern-opisyvaet-problemu-kotoraia-vstrecha/>
13. [Электронный ресурс]. - Режим доступа:
14. [Электронный ресурс]. - Режим доступа:

# ПРИЛОЖЕНИЕ А