Министерство Образования 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 турагентства»

Выполнил: ст. гр. TI-196 Lisnic Andrei

Проверил: унив. лектор Поштару.А

Кишинев, 2022

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....3
1. ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ.....3-4
1.1. История создания шаблонов проектирования.....4-5
1.2. Классификация паттернов.....5
1.3. Порождающие паттерны проектирования.....6
1.4. Структурные паттерны проектирования.....6-7
1.5. Поведенческие паттерны проектирования.....7-8
2. ПРОЕКТИРОВАНИЯ И РАЗРАБОТКА.....8
2.1. **Порождающие паттерны**……....8
2.1.1. Фабричный метод (Factory Method) ……..8-10
2.2.1. Абстрактная фабрика (Abstract Factory)……...10-12
2.2. **Структурные паттерны**…….12
2.2.2. Декораторы (Decorator) ……..……13-15
2.3.2 Заместитель (Proxy) ……..……..15-17
2.3. **Поведенческие паттерны** ……..17
2.3.3. Команда (Command) ……...……18-20
2.4.3. Цепочка Обязанностей (Chain of responsibility) ……20-22
ЗАКЛЮЧЕНИЕ.....23
ИСТОЧНИКИ.....24
ПРИЛОЖЕНИЕ А.....25-27
ПРИЛОЖЕНИЕ В.....27-30
ПРИЛОЖЕНИЕ С.....30-31

Введение

Большинство задач, с которыми часто приходится сталкиваться программистам, уже давным давно решены другими разработчиками. Шаблоны проектирования как раз и являются тем средством, с помощью которого люди могут поделиться друг с другом накопленным опытом. Как только шаблон становится всеобщим достоянием, он обогащает наш язык и позволяет легко поделиться с другими новыми идеями проектирования и их результатами.

С помощью шаблонов проектирования просто выделяют общие задачи, определяют проверенные решения и описывают вероятные результаты. Во многих книгах и статьях описываются особенности конкретных языков программирования, имеющиеся функции, классы и методы. А в каталогах шаблонов, наоборот, упор сделан на том, как перейти в ваших проектах от этих основ (“что именно”) к пониманию задач и возможных решений (“почему” и “как”).

С помощью шаблонов проектирования описываются и формализуются типовые задачи и их решения. В результате опыт, который нарабатывается с большим тру дом, становится доступным широкому сообществу программистов. Шаблоны должны быть построены, главным образом, по “восходящему”, а не “нисходящему” принципу. Их корень — в практике, а не в теории. Но это совсем не означает, что в шаблонах проектирования отсутствует элемент теории. Шаблоны основаны на реальных методах, используемых реальными программистами.

В данной работе, было написано приложение, с помощью которого можно зарегистрировать пользователей и выбрать тур в который они хотят отправиться.

1.ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Шаблоны проектирования представляют собой лучшие практики, используемые опытными разработчиками объектно-ориентированного программного обеспечения. Шаблоны проектирования — это решения общих проблем, с которыми разработчики программного обеспечения столкнулись во время разработки программного обеспечения. Эти решения были получены методом проб и ошибок многочисленными разработчиками программного обеспечения в течение довольно значительного периода времени.

Шаблоны проектирования могут ускорить процесс разработки, предоставляя проверенные и проверенные парадигмы разработки. Эффективная разработка программного обеспечения требует рассмотрения проблем, которые могут стать видимыми лишь на более позднем этапе

реализации. Повторное использование шаблонов проектирования помогает предотвратить тонкие проблемы, которые могут вызвать серьезные проблемы, и улучшает читаемость кода для программистов и архитекторов, знакомых с шаблонами.

Кроме того, шаблоны позволяют разработчикам общаться, используя хорошо известные и понятные имена для взаимодействия с программным обеспечением. Общие шаблоны проектирования могут быть улучшены с течением времени, что сделает их более надежными, чем специальные проекты.

1.1 История создания шаблонов проектирования

Шаблоны проектирования восходят к концу 1970-х годов, когда была опубликована *книга «Язык шаблонов: города, здания, строительство* » архитектора Кристофера Александра и некоторых других. Эта книга представила шаблоны проектирования в архитектурном контексте, представив 253 шаблона, которые в совокупности сформировали то, что авторы назвали *языком шаблонов* .

Концепция языка шаблонов впоследствии появилась в книге Дональда Нормана и Стивена Дрейпера « *Проектирование систем, ориентированных на пользователя»* , которая была опубликована в 1986 году. В этой книге предлагалось применять языки шаблонов в *интерактивном дизайне* , который является практикой проектирования интерактивных цифровых продуктов, сред, систем. и сервисы для использования человеком.

Тем временем Кент Бек и Уорд Каннингем начали изучать шаблоны и их применимость к разработке программного обеспечения. В 1987 году они использовали серию шаблонов проектирования, чтобы помочь Tektronix's Semiconductor Test Systems Group, у которой возникли проблемы с завершением проекта. Бек и Каннингем последовали совету Александра по дизайну, ориентированному на пользователя (позволив представителям пользователей проекта определять результат проектирования), а также предоставили им некоторые шаблоны проектирования, облегчающие работу.

Эрих Гамма также осознал важность повторяющихся шаблонов проектирования во время работы над своей докторской диссертацией. Он считал, что шаблоны проектирования могут облегчить задачу написания повторно используемого объектно-ориентированного программного обеспечения, и размышлял о том, как их документировать и эффективно передавать. Перед Европейской конференцией по объектно-ориентированному программированию 1991 года Гамма и Ричард Хелм начали каталогизировать шаблоны.

На семинаре OOPSLA, проведенном в 1991 году, к Гамме и Хелму присоединились Ральф Джонсон и Джон Влиссидес. Эта *«банда четырех»* (GoF), как их впоследствии стали называть, написала популярную *книгу Design Patterns: Elements of Reusable Object-Oriented Software* , которая документирует 23 шаблона проектирования в трех категориях.

1.2 Классификация паттернов

Шаблоны проектирования в основном подразделяются на три категории: **порождающий шаблон проектирования, структурный шаблон проектирования и поведенческий шаблон проектирования.** Они отличаются друг от друга уровнем детализации, сложности и масштабом применимости ко всей разрабатываемой системе.

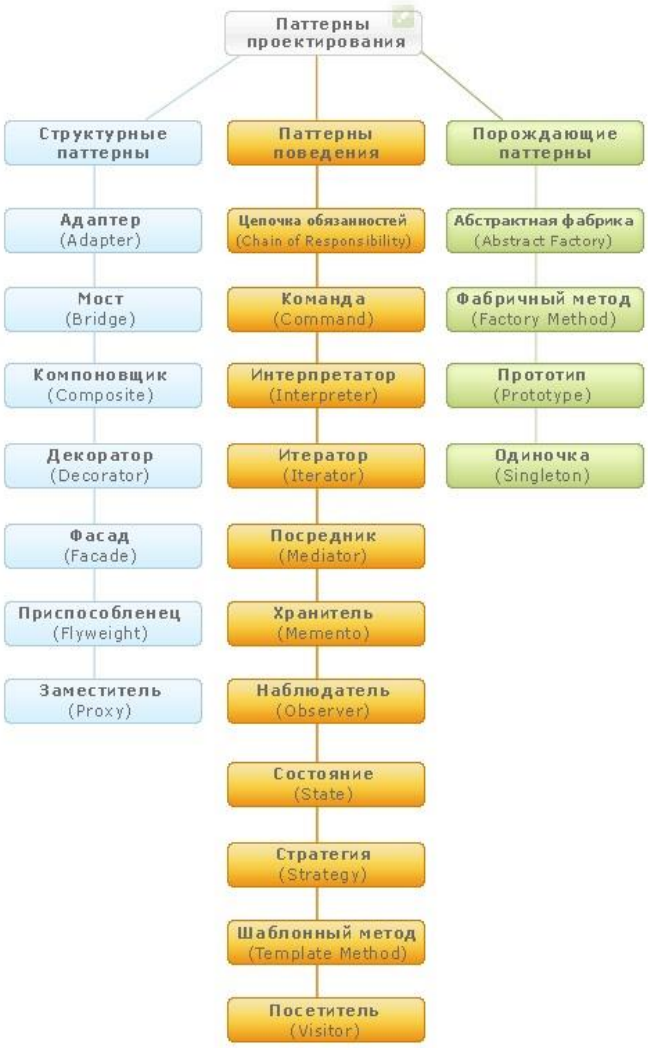


Рисунок 1. – Классификация паттернов

1.3 Порождающие паттерны проектирования

В разработке программного обеспечения порождающие шаблоны проектирования — это шаблоны проектирования, которые имеют дело с механизмами создания объектов, пытаясь

создать объекты способом, подходящим для ситуации. Базовая форма создания объекта может привести к проблемам с дизайном или усложнить дизайн. Порождающие шаблоны проектирования решают эту проблему, каким-то образом контролируя создание этого объекта.

- **Abstract Factory(Абстрактная фабрика)**
  Создает экземпляр нескольких семейств классов
- **Builder(Строитель)**
  Отделяет построение объекта от его представления
- **Factory Method(Фабричный метод)**
  Создает экземпляр нескольких производных классов
- **Ojbect Pool(Пул объектов)**
  Избегайте дорогостоящего приобретения и высвобождения ресурсов за счет утилизации объектов, которые больше не используются.
- **Prototype(Прототип)**
  Полностью инициализированный экземпляр для копирования или клонирования.
- **Singleton(Одиночка)**
  Класс, у которого может существовать только один экземпляр.

1.4 Структурные паттерны проектирования

В программной инженерии шаблоны структурного проектирования — это шаблоны проектирования, которые упрощают проектирование, определяя простой способ реализации взаимосвязей между объектами.

- **Adapter(Адаптер)**
  Сопоставьте интерфейсы разных классов
- **Brige(Мост)**
  Отделяет интерфейс объекта от его реализации.
- **Composite(Составной)**
  Древовидная структура простых и составных объектов.
- **Decorator(Декоратор)**
  Динамическое добавление обязанностей к объектам
- **Facade(Фасад)**
  Один класс, представляющий всю подсистему.

- **Flyweight(Приспособленец**)

  Детализированный экземпляр, используемый для эффективного совместного использования
- **Proxy(Прокси)**

  Объект, представляющий другой объект

1.5 Поведенческие паттерны проектирования

В программной инженерии шаблоны поведенческого проектирования — это шаблоны проектирования, которые определяют общие шаблоны связи между объектами и реализуют эти шаблоны. Поступая таким образом, эти шаблоны увеличивают гибкость в осуществлении этого общения.

- **Chain-of-responsibility(Цепочка-ответственности)**

  Способ передачи запроса между цепочкой объектов
- **Command(Команда)**

  Инкапсулировать запрос команды как объект
- **Interpreter(Интерпретатор)**

  Способ включения языковых элементов в программу.
- **Iterator(Итератор)**

  Последовательный доступ к элементам коллекции
- **Mediator(Медиатор)**

  Определяет упрощенную связь между классами
- **Memento**

  Захват и восстановление внутреннего состояния объекта
- **Null-Object**

  Разработан, чтобы действовать как значение объекта по умолчанию.
- **Observe(Наблюдатель)**

  Способ уведомления об изменении ряда классов.
- **State(Состояние)**

  Изменение поведения объекта при изменении его состояния

- **Strategy(Стратегия)**

  Инкапсулирует алгоритм внутри класса
- **Template-method(Метод-шаблона)**

  Отложите точные шаги алгоритма до подкласса
- **Visitor(Посетитель)**

  Определяет новую операцию для класса без изменений

2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА

В данном приложении, в качестве языка программирования был выбран JavaScript. В качестве среды разработки был использован редактор WebStrom.

2.1. Порождающий паттерн

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 инстанцирование другому объекту.

2.1.1 Фабричный метод (Factory Method)

Фабричный метод — это порождающий шаблон проектирования , т. е. связанный с созданием объекта. В шаблоне Factory мы создаем объекты, не раскрывая логику создания клиенту, и клиент использует тот же общий интерфейс для создания нового типа объекта.
Идея состоит в том, чтобы использовать статическую функцию-член (статический фабричный метод), которая создает и возвращает экземпляры, скрывая от пользователя детали модулей класса.

Фабричный шаблон — это один из основных принципов проектирования для создания объекта, позволяющий клиентам создавать объекты библиотеки (поясняется ниже) таким образом, чтобы они не были тесно связаны с иерархией классов библиотеки.

Фабричный метод используется в следующих ситуациях:

- Когда заранее неизвестно, объекты каких типов необходимо создавать

- Когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать.
- Когда создание новых объектов необходимо делегировать из базового класса классам наследникам

На языке UML паттерн можно описать следующим образом:

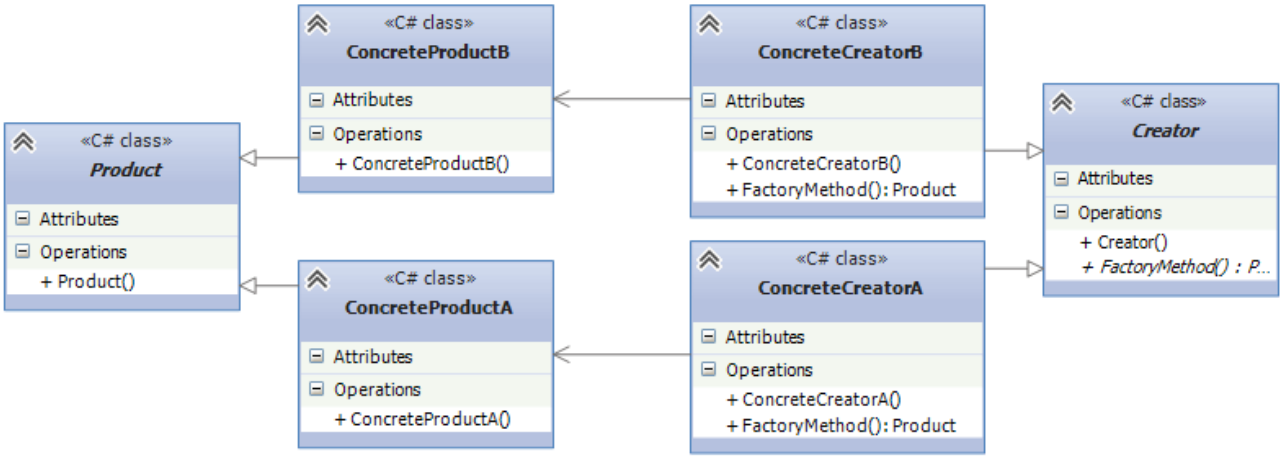


Рисунок 2. – Диаграмма Фабричного метода

Участники:

- Абстрактный класс **Product** определяет интерфейс класса, объекты которого надо создавать.
- Конкретные классы **ConcreteProductA** и **ConcreteProductB** представляют реализацию класса Product. Таких классов может быть множество
- Абстрактный класс **Creator** определяет абстрактный фабричный метод FactoryMethod(), который возвращает объект Product.
- Конкретные классы **ConcreteCreatorA** и **ConcreteCreatorB** - наследники класса Creator, определяющие свою реализацию метода FactoryMethod(). Причем метод FactoryMethod() каждого отдельного класса-создателя возвращает определенный конкретный тип продукта. Для каждого конкретного класса продукта определяется свой конкретный класс создателя.

Таким образом, класс Creator делегирует создание объекта Product своим наследникам. А классы ConcreteCreatorA и ConcreteCreatorB могут самостоятельно выбирать какой конкретный тип продукта им создавать.

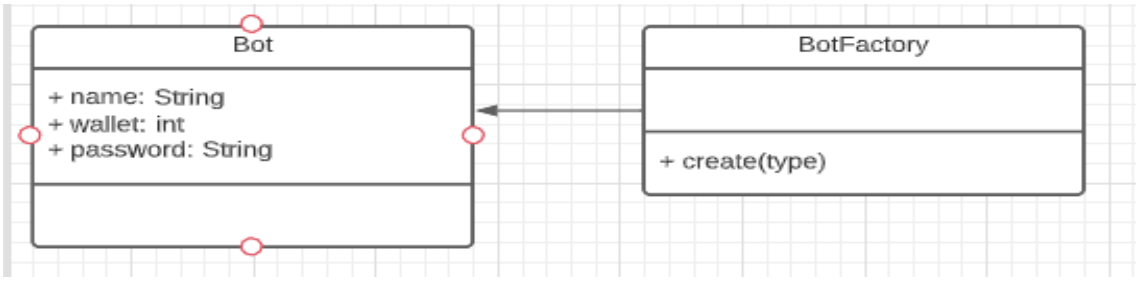


Рисунок 3. Диаграмма Фабричного метода проекта.

На рисунки 3, можно увидеть два созданных класса Bot и BotFactory с помощью которых мы можем создать несколько пользователей.

2.2.1 Абстрактная фабрика (Abstract Factory)

Шаблон проектирования «Абстрактная фабрика» является одним из шаблонов создания. Абстрактный фабричный паттерн почти аналогичен фабричному паттерну и рассматривается как еще один уровень абстракции над фабричным паттерном. Шаблоны абстрактных фабрик работают вокруг суперфабрики, которая создает другие фабрики. Реализация шаблона абстрактной фабрики предоставляет нам структуру, позволяющую создавать объекты, соответствующие общему шаблону. Таким образом, во время выполнения абстрактная фабрика соединяется с любой желаемой конкретной фабрикой, которая может создавать объекты желаемого типа.

Абстрактная фабрика используется в следующих ситуациях:

- Когда система не должна зависеть от способа создания и компоновки новых объектов
- Когда создаваемые объекты должны использоваться вместе и являются взаимосвязанными

На языке UML паттерн можно описать следующим образом:

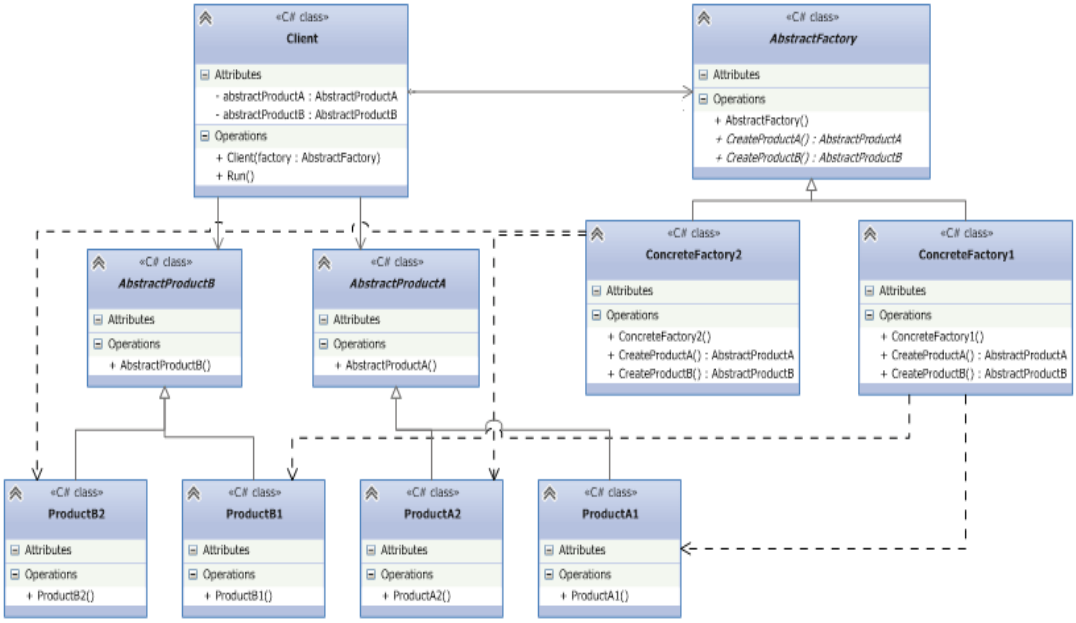


Рисунок 4. – Диаграмма Абстрактной фабрики

Участники:

- Абстрактные классы **AbstractProductA** и **AbstractProductB** определяют интерфейс для классов, объекты которых будут создаваться в программе.
- Конкретные классы **ProductA1** / **ProductA2** и **ProductB1** / **ProductB2** представляют конкретную реализацию абстрактных классов
- Абстрактный класс фабрики **AbstractFactory** определяет методы для создания объектов. Причем методы возвращают абстрактные продукты, а не их конкретные реализации.
- Конкретные классы фабрик **ConcreteFactory1** и **ConcreteFactory2** реализуют абстрактные методы базового класса и непосредственно определяют какие конкретные продукты использовать
- Класс клиента **Client** использует класс фабрики для создания объектов. При этом он использует исключительно абстрактный класс фабрики AbstractFactory и абстрактные классы продуктов AbstractProductA и AbstractProductB и никак не зависит от их конкретных реализаций

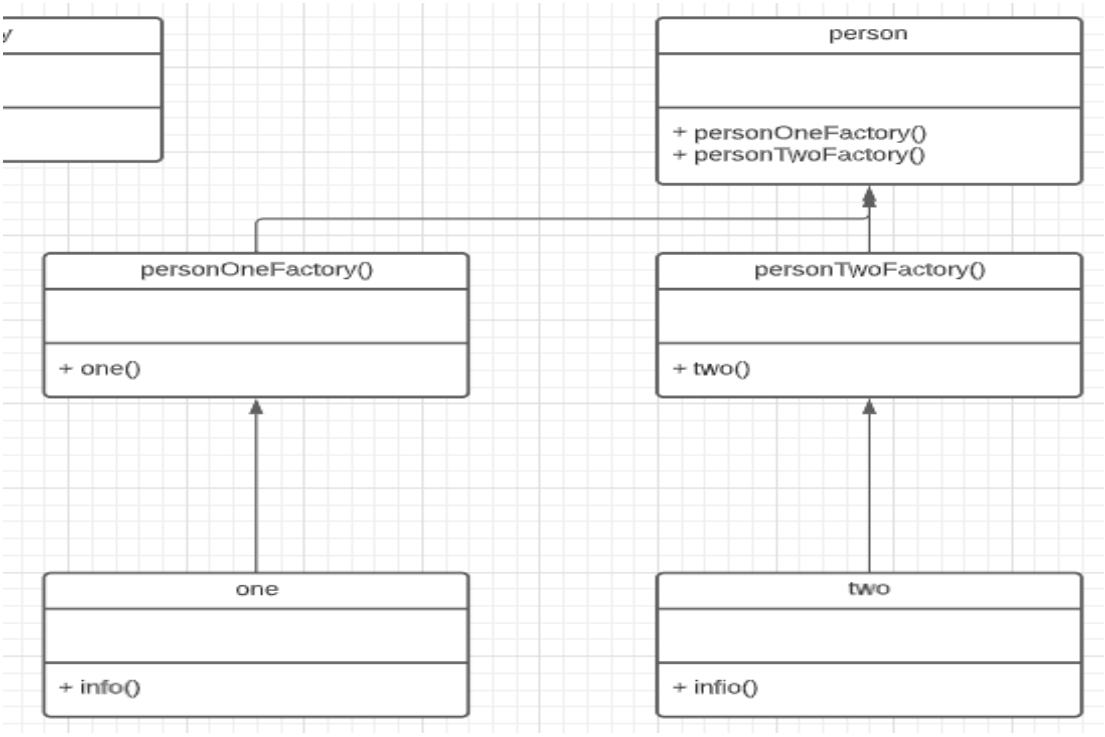


Рисунок 5. – Диаграмма Абстрактной фабрики проекта

На рисунке 5 можно увидеть главный класс person, который возвращает выбранного нами пользователя. Так же были созданы два класса personOneFactory и personTwoFactory которые наследуются от класса person, и который возвращают результат двух других классов one и two которые так же наследуются.

2.2. **Структурные паттерны**

Структурный шаблон проектирования – это план того , как различные объекты и классы объединяются вместе, чтобы сформировать более крупную структуру для достижения нескольких целей в целом. Шаблоны структурных проектах показывают, как уникальные части системы могут быть объединены вместе расширяемым и гибким образом. Итак, с помощью шаблона структурного проектирования мы можем нацеливаться и изменять определенные части структуры, не изменяя всей структуры.

2.2.2. Декораторы (Decorator)

Шаблон проектирования Decorator позволяет нам динамически добавлять функциональность и поведение к объекту, не влияя на поведение других существующих объектов в том же классе. Мы используем наследование для расширения поведения класса. Это происходит во время компиляции, и все экземпляры этого класса получают расширенное поведение.

- Шаблоны декораторов позволяют пользователю добавлять новые функции к существующему объекту, не изменяя его структуру. Таким образом, в исходном классе нет никаких изменений.
- Шаблон проектирования декоратора — это структурный шаблон, который обеспечивает оболочку для существующего класса.
- Шаблон проектирования Decorator использует абстрактные классы или интерфейсы с композицией для реализации оболочки.
- Шаблоны проектирования декораторов создают классы декораторов, которые обертывают исходный класс и обеспечивают дополнительную функциональность, сохраняя неизменной сигнатуру методов класса.
- Шаблоны проектирования декораторов чаще всего используются для применения принципов единой ответственности, поскольку мы разделяем функциональность на классы с уникальными областями ответственности.
- Шаблон проектирования декоратора структурно почти подобен шаблону цепочки ответственности.

Декоратор используется в следующих ситуациях:

- Когда надо динамически добавлять к объекту новые функциональные возможности. При этом данные возможности могут быть сняты с объекта
- Когда применение наследования неприемлемо. Например, если нам надо определить множество различных функциональностей и для каждой функциональности наследовать отдельный класс, то структура классов может очень сильно разрастись. Еще больше она может разрастись, если нам необходимо создать классы, реализующие все возможные сочетания добавляемых функциональностей.

На языке UML паттерн можно описать следующим образом:

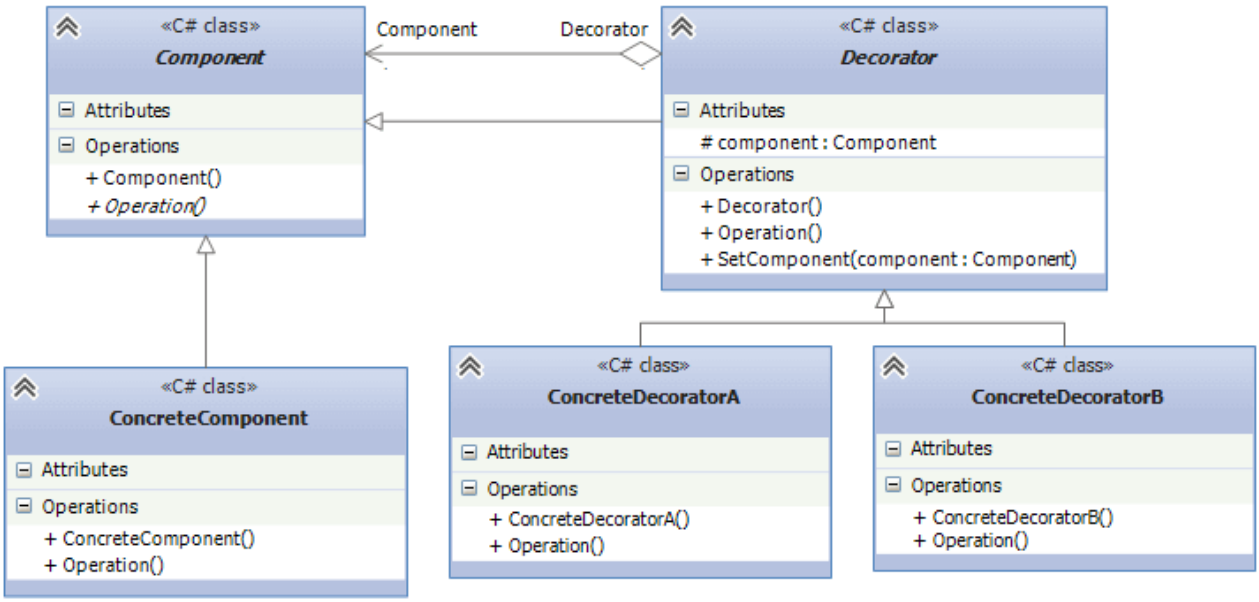


Рисунок 6. – Диаграмма Декоратора

Участники

- **Component**: абстрактный класс, который определяет интерфейс для наследуемых объектов
- **ConcreteComponent**: конкретная реализация компонента, в которую с помощью декоратора добавляется новая функциональность
- **Decorator**: собственно декоратор, реализуется в виде абстрактного класса и имеет тот же базовый класс, что и декорируемые объекты. Поэтому базовый класс Component должен быть по возможности легким и определять только базовый интерфейс.

  Класс декоратора также хранит ссылку на декорируемый объект в виде объекта базового класса Component и реализует связь с базовым классом как через наследование, так и через отношение агрегации.

- Классы **ConcreteDecoratorA** и **ConcreteDecoratorB** представляют дополнительные функциональности, которыми должен быть расширен объект ConcreteComponent

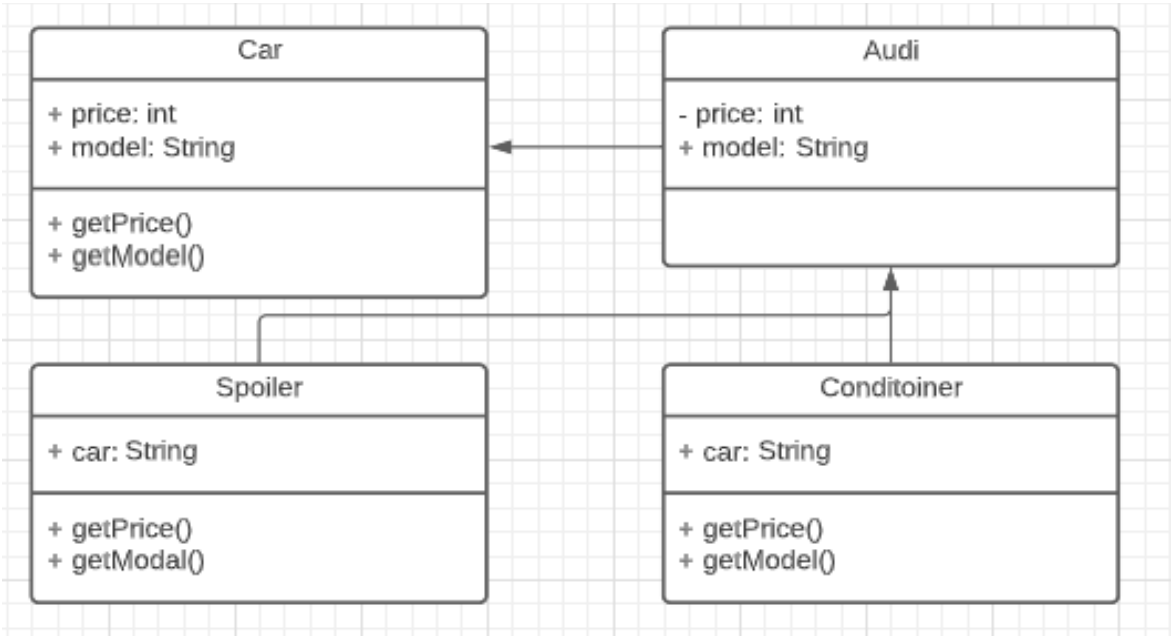


Рисунок 7. – Диаграмма Декоратора проекта

На рисунке 7, можно увидеть главный класс Car, который создаёт цены машины и модель и возвращает их. Так же есть определённый класс Audi который наследуется от класса Car, где задали определённый автомобиль. Далее у нас идут два декоратора это класс Spoiler и Conditioner.

2.3.2 Заместитель (Proxy)

Прокси означает «вместо», «представлять», «вместо» или «от имени» и является буквальным значением прокси, и это напрямую объясняет **шаблон проектирования прокси** . Прокси также называют суррогатами, дескрипторами и обертками. Они тесно связаны по структуре, но не по назначению, с адаптерами и декораторами .

Примером из реального мира может быть чек или кредитная карта, которые являются прокси того, что находится на нашем банковском счете. Он может использоваться вместо наличных денег и обеспечивает средства доступа к этим наличным деньгам, когда это необходимо. И это именно то, что делает паттерн Proxy — « **Контролирует и управляет доступом к объекту, который они защищают** ».

Прокси используется в следующих ситуациях:

- Когда надо осуществлять взаимодействие по сети, а объект-проси должен имитировать поведения объекта в другом адресном пространстве. Использование прокси позволяет

снизить накладные издержки при передачи данных через сеть. Подобная ситуация еще называется **удалённый заместитель (remote proxies)**

- Когда нужно управлять доступом к ресурсу, создание которого требует больших затрат. Реальный объект создается только тогда, когда он действительно может понадобится, а до этого все запросы к нему обрабатывает прокси-объект. Подобная ситуация еще называется **виртуальный заместитель (virtual proxies)**
- Когда необходимо разграничить доступ к вызываемому объекту в зависимости от прав вызывающего объекта. Подобная ситуация еще называется **защищающий заместитель (protection proxies)**
- Когда нужно вести подсчет ссылок на объект или обеспечить потокобезопасную работу с реальным объектом. Подобная ситуация называется **"умные ссылки" (smart reference)**

На языке UML паттерн можно описать следующим образом:

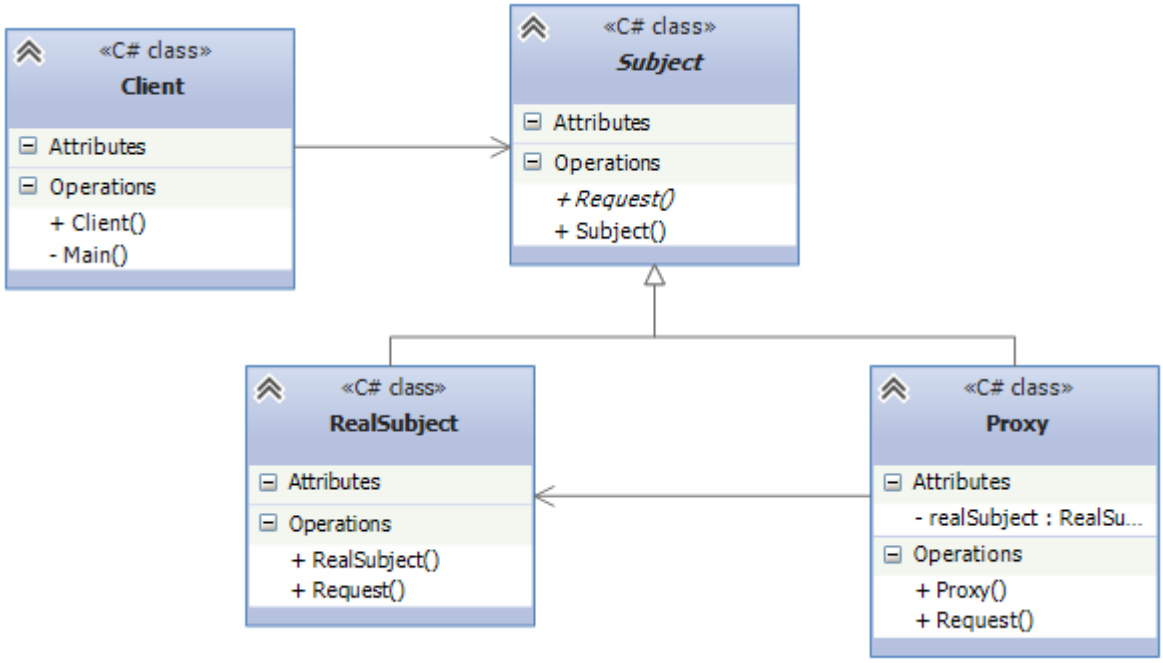


Рисунок 8.- Диаграмма прокси

Участники:

- **Subject**: определяет общий интерфейс для Proxy и RealSubject. Поэтому Proxy может использоваться вместо RealSubject
- **RealSubject**: представляет реальный объект, для которого создается прокси

- **Proxy**: заместитель реального объекта. Хранит ссылку на реальный объект, контролирует к нему доступ, может управлять его созданием и удалением. При необходимости Proxy переадресует запросы объекту RealSubject
- **Client**: использует объект Proxy для доступа к объекту RealSubject

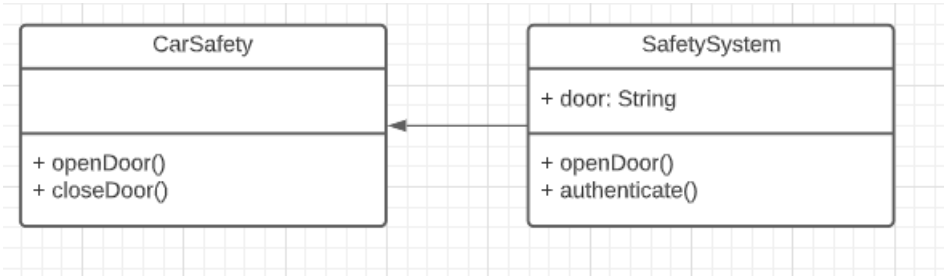


Рисунок 9. – Диаграмма прокси проекта

На рисунке 9, можно увидеть два класса. CarSafety отвечает за открытие и закрытие двери, то есть с помощью данного паттерна была реализована проверка на безопасность открытие двери, как раз второй класс за это и отвечает.

2.2. **Поведенческие паттерны**

Шаблон *поведения* абстрагирует действие, которое вы хотите выполнить, от объекта или класса, который выполняет это действие. Изменяя объект или класс, вы можете изменить используемый *алгоритм* , затрагиваемые объекты или поведение, сохраняя при этом тот же базовый-интерфейс-для *клиентских* классов.

Хороший набор поведенческих паттернов позволяет решать многие сложные проблемы, с которыми вы, вероятно, столкнетесь при проектировании объектно-ориентированных систем. К ним относятся перечисление списков, реагирование на изменения состояния объекта, сериализация и десериализация объектов без проникновения в инкапсуляцию данных.

2.3.3. Команда (Command)

Шаблон **команды** инкапсулирует запрос в виде объекта, что позволяет нам параметризовать другие объекты с помощью различных запросов, ставить в очередь или регистрировать запросы, а также поддерживать операции отмены.

Поначалу это определение немного сбивает с толку, но давайте рассмотрим его. По аналогии с нашей проблемой выше, дистанционное управление является клиентом, а стереосистема, свет и т. д. — приемниками. В шаблоне команды есть объект Command, который *инкапсулирует запрос* , связывая вместе набор действий на конкретном получателе. Он делает это, предоставляя только один метод execute(), который вызывает некоторые действия для получателя.

Параметризация других объектов с другими запросами в нашей аналогии означает, что кнопка, используемая для включения света, может впоследствии использоваться для включения стереосистемы или, возможно, для открытия двери гаража.

ставить в очередь или регистрировать запросы, а также поддерживать отмену операций означает, что операция Execute команды может сохранять состояние для отмены ее эффектов в самой команде.

Команда может иметь добавленную операцию unExecute, которая отменяет эффекты предыдущего вызова для выполнения. Она также может поддерживать регистрацию изменений, чтобы их можно было повторно применить в случае сбоя системы.

Команда используется в следующих ситуациях:

- Когда надо передавать в качестве параметров определенные действия, вызываемые в ответ на другие действия. То есть когда необходимы функции обратного действия в ответ на определенные действия.
- Когда необходимо обеспечить выполнение очереди запросов, а также их возможную отмену.
- Когда надо поддерживать логгирование изменений в результате запросов. Использование логов может помочь восстановить состояние системы - для этого необходимо будет использовать последовательность запротоколированных команд.

На языке UML паттерн можно описать следующим образом:

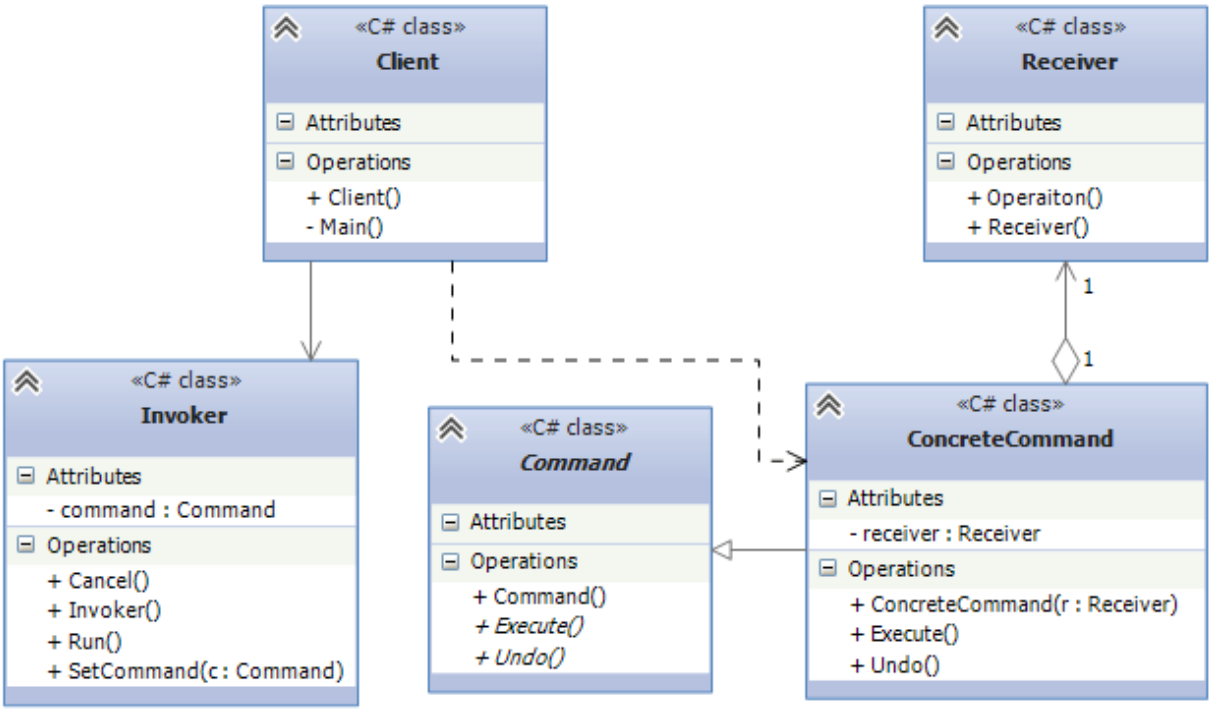


Рисунок 10. – Диаграмма Команды

Участники:

- **Command**: интерфейс, представляющий команду. Обычно определяет метод Execute() для выполнения действия, а также нередко включает метод Undo(), реализация которого должна заключаться в отмене действия команды
- **ConcreteCommand**: конкретная реализация команды, реализует метод Execute(), в котором вызывается определенный метод, определенный в классе Receiver
- **Receiver**: получатель команды. Определяет действия, которые должны выполняться в результате запроса.
- **Invoker**: инициатор команды - вызывает команду для выполнения определенного запроса
- **Client**: клиент - создает команду и устанавливает ее получателя с помощью метода SetCommand()

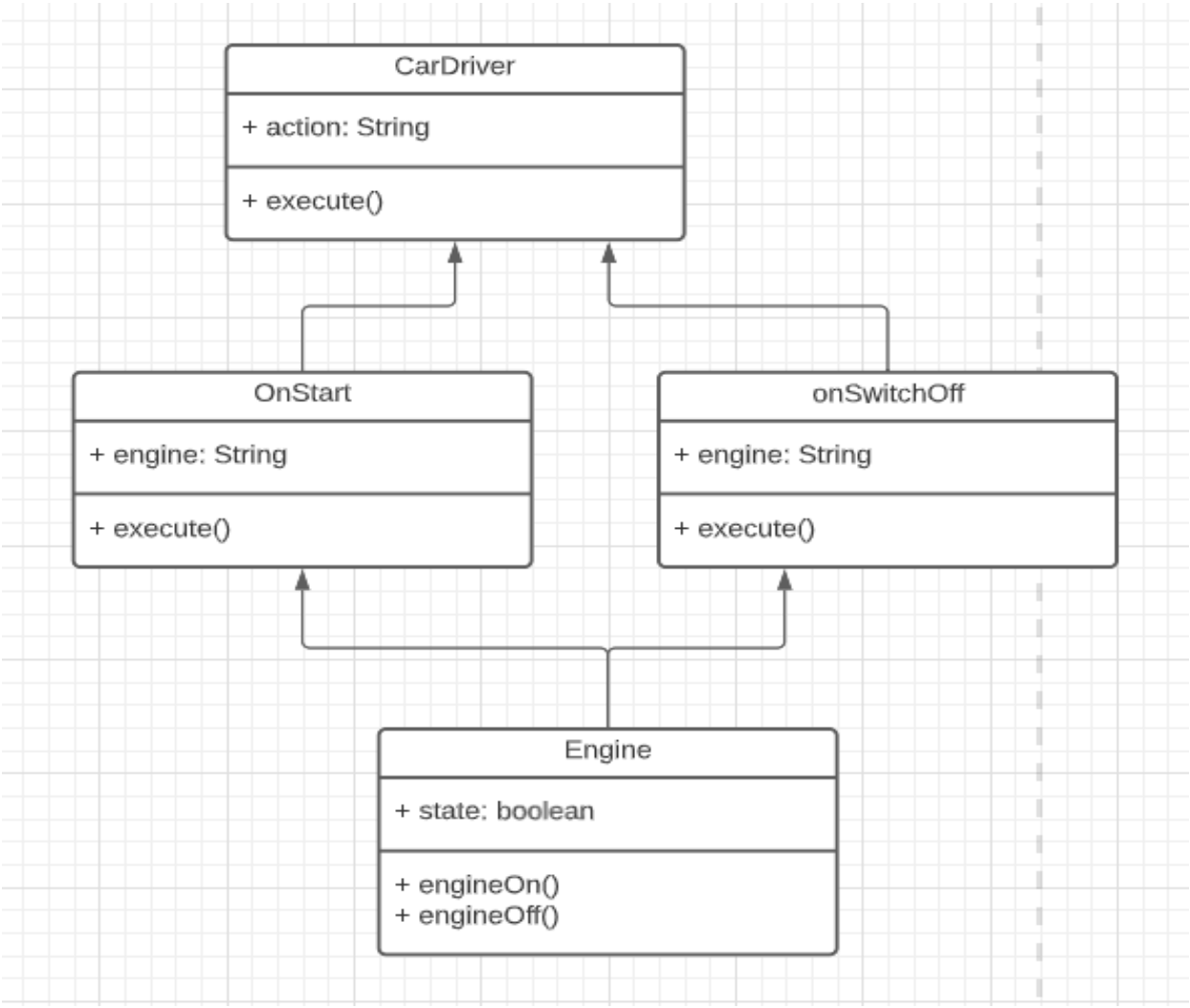


Рисунок 11. – Диаграмма Команды проекта

На рисунке 11, показана реализация проверки включения и выключения автомобиля, где есть класс CarDriver, который не может на прямую взаимодействовать с двигателем, из за этого созданы были два класса onStart и onSwitchOff как ключ автомобиля , с помощью которого можно завести автомобиль.

2.4.3. **Цепочка Обязанностей (Chain of responsibility)**

Шаблон цепочки ответственности используется для достижения слабой связи в разработке программного обеспечения, когда запрос от клиента передается цепочке объектов для их обработки. Позже объект в цепочке сам решит, кто будет обрабатывать запрос и требуется ли отправить запрос следующему объекту в цепочке или нет.

Цепочка Обязанностей используется в следующих ситуациях:

- Когда вы хотите разделить отправителя и получателя запроса
- Несколько объектов, определенных во время выполнения, являются кандидатами на обработку запроса.
- Когда вы не хотите явно указывать обработчики в своем коде
- Когда вы хотите сделать запрос к одному из нескольких объектов без явного указания получателя.

На языке UML паттерн можно описать следующим образом:

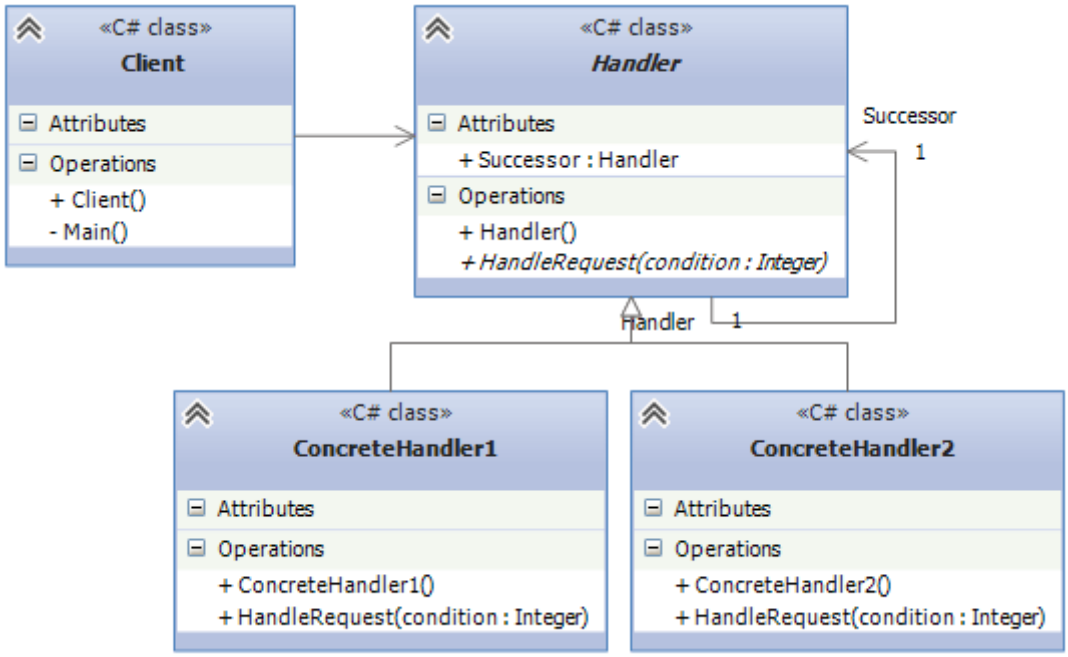


Рисунок 12. – Диаграмма Цепочки обязанностей

Участники:

- **Handler**: определяет интерфейс для обработки запроса. Также может определять ссылку на следующий обработчик запроса
- **ConcreteHandler1** и **ConcreteHandler2**: конкретные обработчики, которые реализуют функционал для обработки запроса. При невозможности обработки и наличия ссылки на следующий обработчик, передают запрос этому обработчику

В данном случае для простоты примера в качестве параметра передается некоторое число, сначала обработчик обрабатывает запрос и, если передано соответствующее число, завершает его обработку. Иначе передает запрос на обработку следующем в цепи обработчику при его наличии.

- **Client**: отправляет запрос объекту Handler

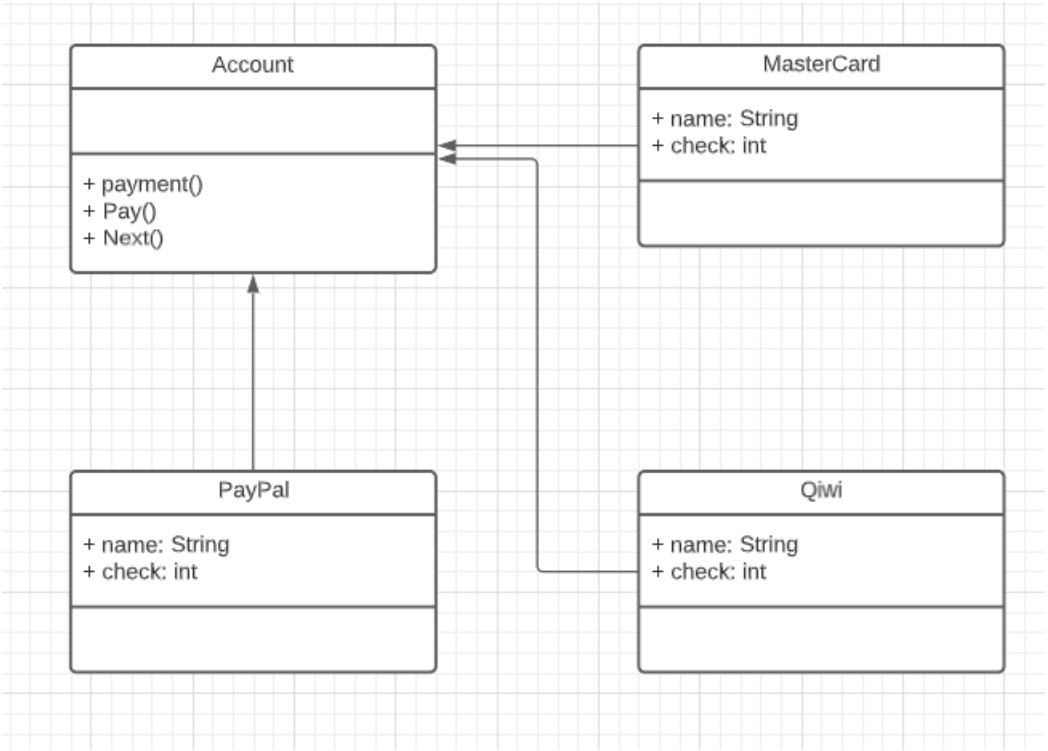


Рисунок 12. – Диаграмма Цепочки обязанностей проекта

На рисунке 12, реализована диаграмма способа оплата с разных кошельков. Есть главный класс Account от которого наследуется три класса MasterCard,PayPal,Qiwi.

Заключения

Паттерны проектирования — это решения распространенных проблем при разработке кода. Их знание и использование позволяет экономить время, используя готовые решения, стандартизировать код и повысить общий словарь.

В зависимости от того, какие задачи решают паттерны проектирования, они **делятся на три вида**: порождающие, структурные и поведенческие.

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.

Рассмотрены одни из самых популярных шаблонов проектирования, которые предоставляют не набор готовых решений, а варианты и подходы к решению распространенных задач, они не зависят от языка программирования и предоставляют лишь готовые абстракции.

С помощью шаблонов проектирования описываются лучшие методики решения задач.

Шаблоны проектирования способствуют повторному использованию кода. Чаще всего, код, написанный с использованием шаблонов проектирования - понятней и легче расширяется.

ИСТОЧНИКИ

1. Паттерны проектирования, польза паттернов, классификация[Электронный ресурс] https://refactoring.guru/ru/design-patterns/creational-patterns

2. Шаблоны проектирования[Электронный ресурс]https://www.geeksforgeeks.org/flyweight-design-pattern/?ref=lbp

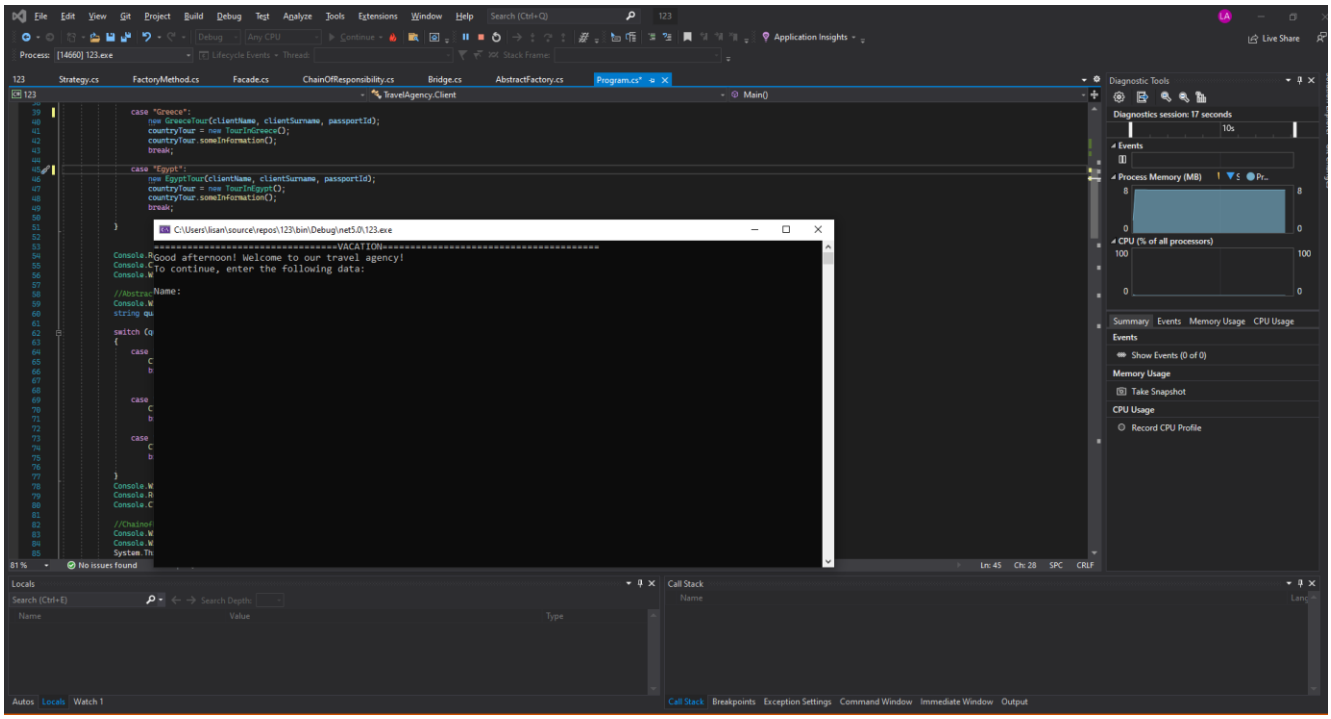
3. UML диаграммы [Электронный ресур]https://evergreens.com.ua/ru/articles/uml-diagrams.html

4. Паттерны проектирования на примере UML [Электронный ресурс] https://intellect.icu/patterny-proektirovaniya-s-primerami-na-uml-diagramme-klassov-7700

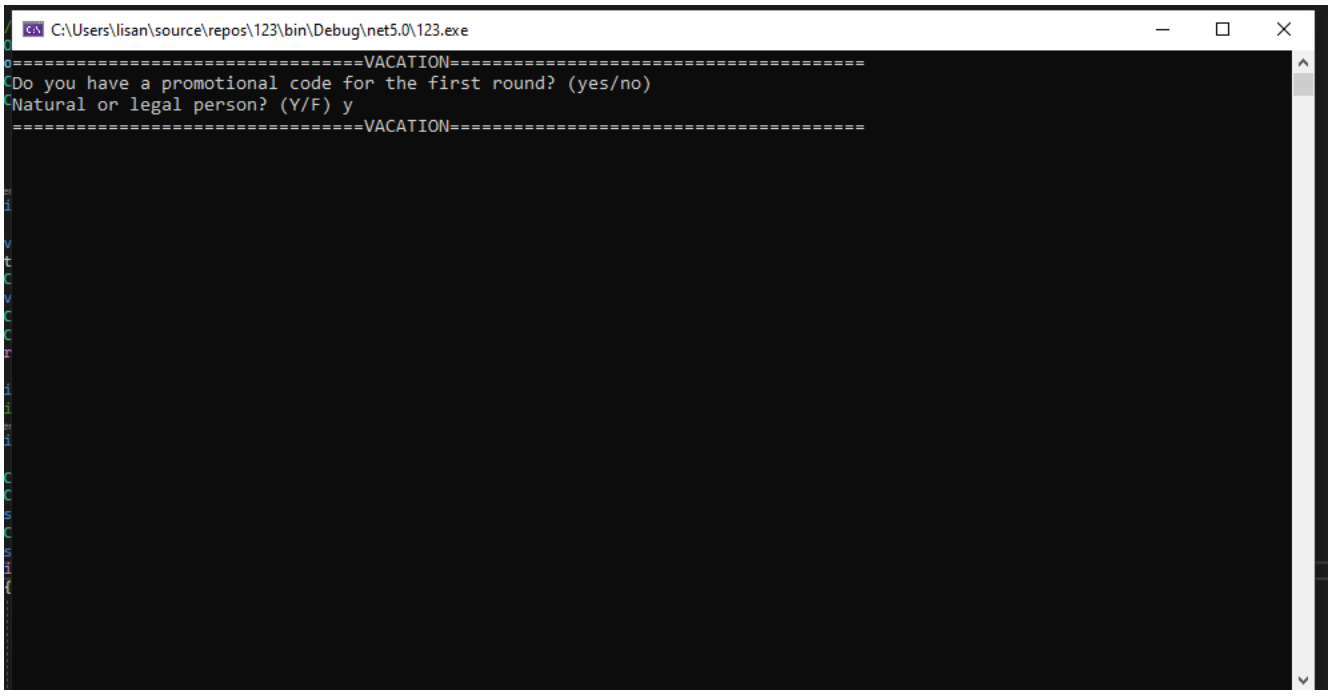
5. Паттерны проектирования [книга] Публикация: октябрь 2004г. Авторы Эрик Фриман, Элизабет Фримен, Берт Бейтс, Кэти Сьерра

ПРИЛОЖЕНИЯ

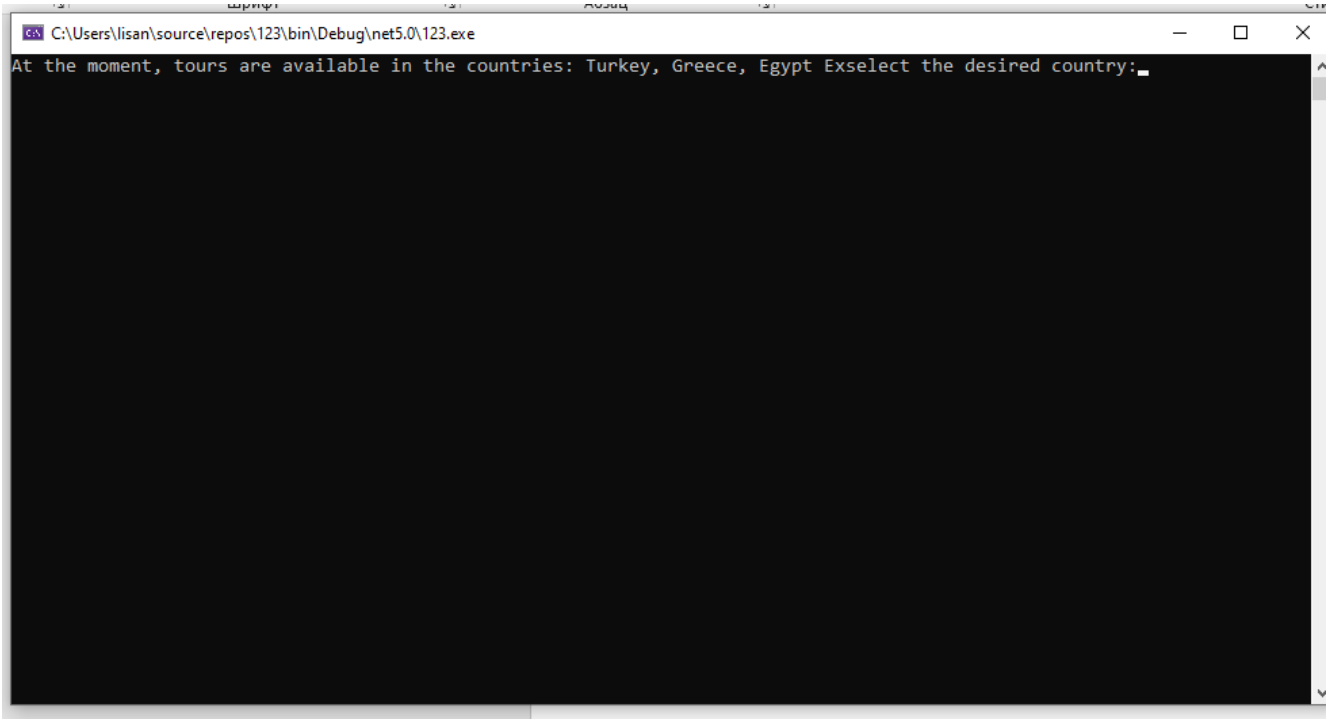
А



В



C



Ссылка на репозиторий github: https://github.com/LisAndrew/tmps