**Лабораторная № 5**

**Применение структурных и паттернов поведения**

1. Назначение структурных паттернов?

**Структурные паттерны** - рассматривает, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты. К таким шаблонам относятся:

* **Адаптер (Adapter)**
* **Мост (Bridge)**
* **Компоновщик (Composite)**
* **Декоратор (Decorator)**
* **Фасад (Facade)**
* **Приспособленец (Flyweight)**
* **Заместитель (Proxy)**



2. Нарисуете диаграмму классов и поясните принцип работы паттерна

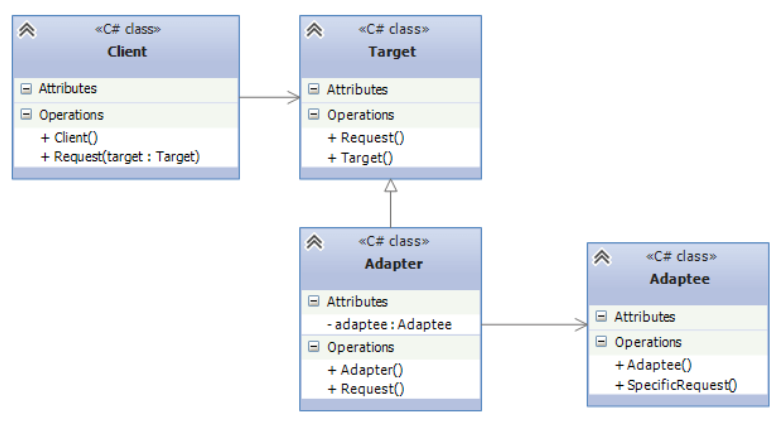
Adapter. В каких случаях надо его применять.

https://metanit.com/sharp/patterns/4.2.php

**Адаптер** — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.

Это объект-переводчик, который трансформирует интерфейс или данные одного объекта в такой вид, чтобы он стал понятен другому объекту.

При этом адаптер оборачивает один из объектов, так что другой объект даже не знает о наличии первого.



Адаптеры могут не только переводить данные из одного формата в другой, но и помогать объектам с разными интерфейсами работать сообща. Это работает так:

1)Адаптер имеет интерфейс, который совместим с одним из объектов.

2)Поэтому этот объект может свободно вызывать методы адаптера.

3)Адаптер получает эти вызовы и перенаправляет их второму объекту, но уже в том формате и последовательности, которые понятны второму объекту.



Когда применять?

* Когда необходимо использовать имеющийся класс, но его интерфейс не соответствует потребностям
* Когда надо использовать уже существующий класс совместно с другими классами, интерфейсы которых не совместимы

 Вы могли бы создать ещё один уровень подклассов и добавить в них недостающую функциональность. Но при этом придётся дублировать один и тот же код в обеих ветках подклассов.

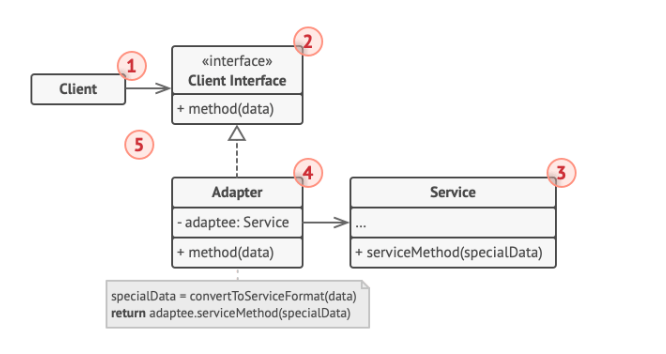
клиент ничего не знает об Adaptee, он знает и использует только объекты Target. И благодаря адаптеру мы можем на клиенте использовать объекты Adaptee как Target

<https://metanit.com/sharp/patterns/4.2.php>

3.В чем разница между адаптером класса и адаптером объекта?

https://refactoring.guru/ru/design-patterns/adapter

* **Адаптер Класс** использует наследование и может обернуть только **класс**. Он не может обернуть интерфейс, так как по определению он должен быть производным от некоторого базового класса. Эта реализация базируется на наследовании: адаптер наследует оба интерфейса одновременно.
* **Объектный Адаптер** использует состав и может обернуть классы или интерфейсы, или оба. Он может сделать это, так как он содержит, как частный, инкапсулированный член, класс или интерфейс **объект** она оборачивается.Эта реализация использует агрегацию: объект адаптера «оборачивает», то есть содержит ссылку на служебный объект.



4. Назначение и принцип организации паттерна Декоратор.

<https://refactoring.guru/ru/design-patterns/decorator>

<https://metanit.com/sharp/patterns/4.1.php>

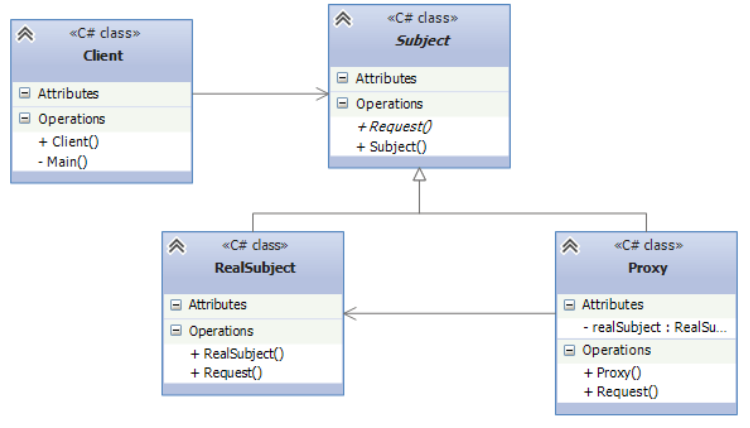
**Декоратор** — это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

**Когда следует использовать декораторы?**

Когда надо динамически добавлять к объекту новые функциональные возможности. При этом данные возможности могут быть сняты с объекта 

5. Нарисуйте диаграмму классов для паттерна Proxy. Поясните его

назначение. Какие разновидности proxy вы знаете?



Заместитель (Proxy) предоставляет объект-заместитель, который управляет доступом к другому объекту. То есть создается объект-суррогат, который может выступать в роли другого объекта и замещать его.

### Когда использовать прокси?

* 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)**

6. В чем разница между паттернами Decorator и Adapter?

**Адаптор**- частный случай декоратора. Отличие скорее в том, что **декоратор**может быть масштабнее (содержать в себе много сущностей, в то время как **адаптор**, как правило, реализует интерфейс для определенного типа).

7. В чем суть паттерна Composite? Поясните в каких случаях его надо

применять?

Компоновщик (Composite) объединяет группы объектов в древовидную структуру по принципу "часть-целое и позволяет клиенту одинаково работать как с отдельными объектами, так и с группой объектов.

бразно реализацию паттерна можно представить в виде меню, которое имеет различные пункты. Эти пункты могут содержать подменю, в которых, в свою очередь, также имеются пункты. То есть пункт меню служит с одной стороны частью меню, а с другой стороны еще одним меню. В итоге мы однообразно можем работать как с пунктом меню, так и со всем меню в целом.

### Когда использовать компоновщик?

* Когда объекты должны быть реализованы в виде иерархической древовидной структуры
* Когда клиенты единообразно должны управлять как целыми объектами, так и их составными частями. То есть целое и его части должны реализовать один и тот же интерфейс

8. Расскажите о паттерне Façade?

Фасад (Facade) представляет шаблон проектирования, который позволяет скрыть сложность системы с помощью предоставления упрощенного интерфейса для взаимодействия с ней.

### Когда использовать фасад?

* Когда имеется сложная система, и необходимо упростить с ней работу. Фасад позволит определить одну точку взаимодействия между клиентом и системой.
* Когда надо уменьшить количество зависимостей между клиентом и сложной системой. Фасадные объекты позволяют отделить, изолировать компоненты системы от клиента и развивать и работать с ними независимо.
* Когда нужно определить подсистемы компонентов в сложной системе. Создание фасадов для компонентов каждой отдельной подсистемы позволит упростить взаимодействие между ними и повысить их независимость друг от друга.

9. В каких случаях надо применять Bridge? Поясните на диаграмме

классов как он применятся

Мост (Bridge) - структурный шаблон проектирования, который позволяет отделить абстракцию от реализации таким образом, чтобы и абстракцию, и реализацию можно было изменять независимо друг от друга.

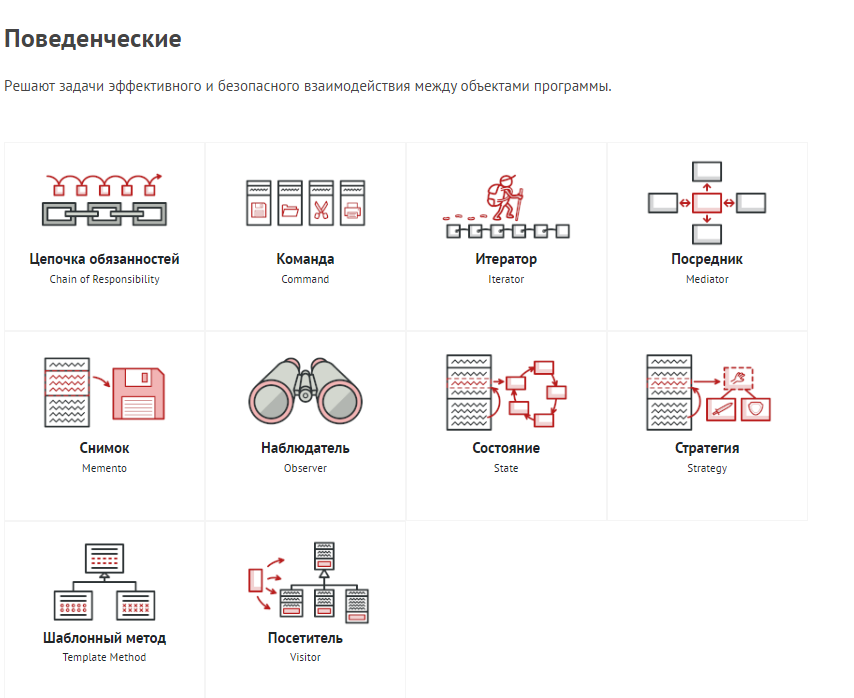
Даже если мы отделим абстракцию от конкретных реализаций, то у нас все равно все наследуемые классы будут жестко привязаны к интерфейсу, определяемому в базовом абстрактном классе. Для преодоления жестких связей и служит паттерн Мост.

### Когда использовать данный паттерн?

* Когда надо избежать постоянной привязки абстракции к реализации
* Когда наряду с реализацией надо изменять и абстракцию независимо друг от друга. То есть изменения в абстракции не должно привести к изменениям в реализации

10. Назначение паттернов поведения?

**Поведенческими** - они определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение.



11. Нарисуете диаграмму классов и поясните принцип работы паттерна

Chain of responsibility. В каких случаях надо его применять?

Цепочка Обязанностей (Chain of responsibility) - поведенческий шаблон проектирования, который позволяет избежать жесткой привязки отправителя запроса к получателю. Все возможные обработчики запроса образуют цепочку, а сам запрос перемещается по этой цепочке. Каждый объект в этой цепочке при получении запроса выбирает, либо закончить обработку запроса, либо передать запрос на обработку следующему по цепочке объекту.

### Когда применяется цепочка обязанностей?

* Когда имеется более одного объекта, который может обработать определенный запрос
* Когда надо передать запрос на выполнение одному из нескольких объект, точно не определяя, какому именно объекту
* Когда набор объектов задается динамически

12. Назначение и принцип организации паттерна Command. Поясните как

он связан с конечными автоматами

**Команда** — это поведенческий паттерн, позволяющий заворачивать запросы или простые операции в отдельные объекты.

Это позволяет откладывать выполнение команд, выстраивать их в очереди, а также хранить историю и делать отмену.

**Применимость:**

- когда нужно откладывать выполнение команд, выстраивать их в очереди, а также хранить историю и делать отмену.

- когда необходимы функции обратного действия в ответ на определенные действия.

- Когда необходимо обеспечить выполнение очереди запросов, а также их возможную отмену.

13. Как реализовать паттерн Observer?

"Наблюдатель" (Observer) представляет поведенческий шаблон проектирования, который использует отношение "один ко многим". В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей.

### Когда использовать паттерн Наблюдатель?

* Когда система состоит из множества классов, объекты которых должны находиться в согласованных состояниях
* Когда общая схема взаимодействия объектов предполагает две стороны: одна рассылает сообщения и является главным, другая получает сообщения и реагирует на них. Отделение логики обеих сторон позволяет их рассматривать независимо и использовать отдельно друга от друга.
* Когда существует один объект, рассылающий сообщения, и множество подписчиков, которые получают сообщения. При этом точное число подписчиков заранее неизвестно и процессе работы программы может изменяться.

14. Нарисуйте диаграмму классов для паттерна Mediator. Поясните его

назначение.

 Посредник (Mediator) представляет такой шаблон проектирования, который обеспечивает взаимодействие множества объектов без необходимости ссылаться друг на друга. Тем самым достигается слабосвязанность взаимодействующих объектов.

Когда используется паттерн Посредник?

* Когда имеется множество взаимосвязаных объектов, связи между которыми сложны и запутаны.
* Когда необходимо повторно использовать объект, однако повторное использование затруднено в силу сильных связей с другими объектами

15. В чем разница между паттернами Mediator и Facade?

Фасад предоставляет существующие функции, а медиатор добавляет к существующим функциям.

16. В чем суть паттерна Memento? Поясните на примере.

 Хранитель (Memento) позволяет выносить внутреннее состояние объекта за его пределы для последующего возможного восстановления объекта без нарушения принципа инкапсуляции.

Когда использовать Memento?

* Когда нужно сохранить состояние объекта для возможного последующего восстановления
* Когда сохранение состояния должно проходить без нарушения принципа инкапсуляции

17. Расскажите о паттерне Visitor?

Посетитель (Visitor) позволяет определить операцию для объектов других классов без изменения этих классов.

При использовании паттерна Посетитель определяются две иерархии классов: одна для элементов, для которых надо определить новую операцию, и вторая иерархия для посетителей, описывающих данную операцию.

Когда использовать данный паттерн?

* Когда имеется много объектов разнородных классов с разными интерфейсами, и требуется выполнить ряд операций над каждым из этих объектов
* Когда классам необходимо добавить одинаковый набор операций без изменения этих классов
* Когда часто добавляются новые операции к классам, при этом общая структура классов стабильна и практически не изменяется

18. В каких случаях надо применять Null object?

**Null Object** — это объект с определенным нейтральным («null») поведением. [Шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) Null Object описывает использование таких объектов и их поведение (или отсутствие такового).

шаблон null object позволяет нам избежать условной сложности, используя объекты, а не примитивные типы.

# Когда использовать

1. Вам нужно добавлять обязанности к отдельным объектам динамически и прозрачно, то есть не затрагивая другие объекты.
2. Вам нужно добавить обязанности, которые могут быть сняты в любой момент.

19. Поясните на диаграмме классов как реализовать Strategy.

Паттерн Стратегия (Strategy) представляет шаблон проектирования, который определяет набор алгоритмов, инкапс улирует каждый из них и обеспечивает их взаимозаменяемость. В зависимости от ситуации мы можем легко заменить один используемый алгоритм другим. При этом замена алгоритма происходит независимо от объекта, который использует данный алгоритм.

### Когда использовать стратегию?

* Когда есть несколько родственных классов, которые отличаются поведением. Можно задать один основной класс, а разные варианты поведения вынести в отдельные классы и при необходимости их применять
* Когда необходимо обеспечить выбор из нескольких вариантов алгоритмов, которые можно легко менять в зависимости от условий
* Когда необходимо менять поведение объектов на стадии выполнения программы
* Когда класс, применяющий определенную функциональность, ничего не должен знать о ее реализации

20. Перечислите и поясните принципы проектирования SOLID.

Принципы:

* **S**ingle Responsibility Principle (Принцип единственной обязанности)
* **O**pen/Closed Principle (Принцип открытости/закрытости)
* **L**iskov Substitution Principle (Принцип подстановки Лисков)
* **I**nterface Segregation Principle (Принцип разделения интерфейсов)
* **D**ependency Inversion Principle (Принцип инверсии зависимостей)

Принципы SOLID - это не паттерны, их нельзя назвать какими-то определенными догмами, которые надо обязательно применять при разработке, однако их использование позволит улучшить код программы, упростить возможные его изменения и поддержку.