1. **Назначение паттернов поведения?**

Паттерны поведения используются для организации, управления и объединения различных вариантов поведения объектов.

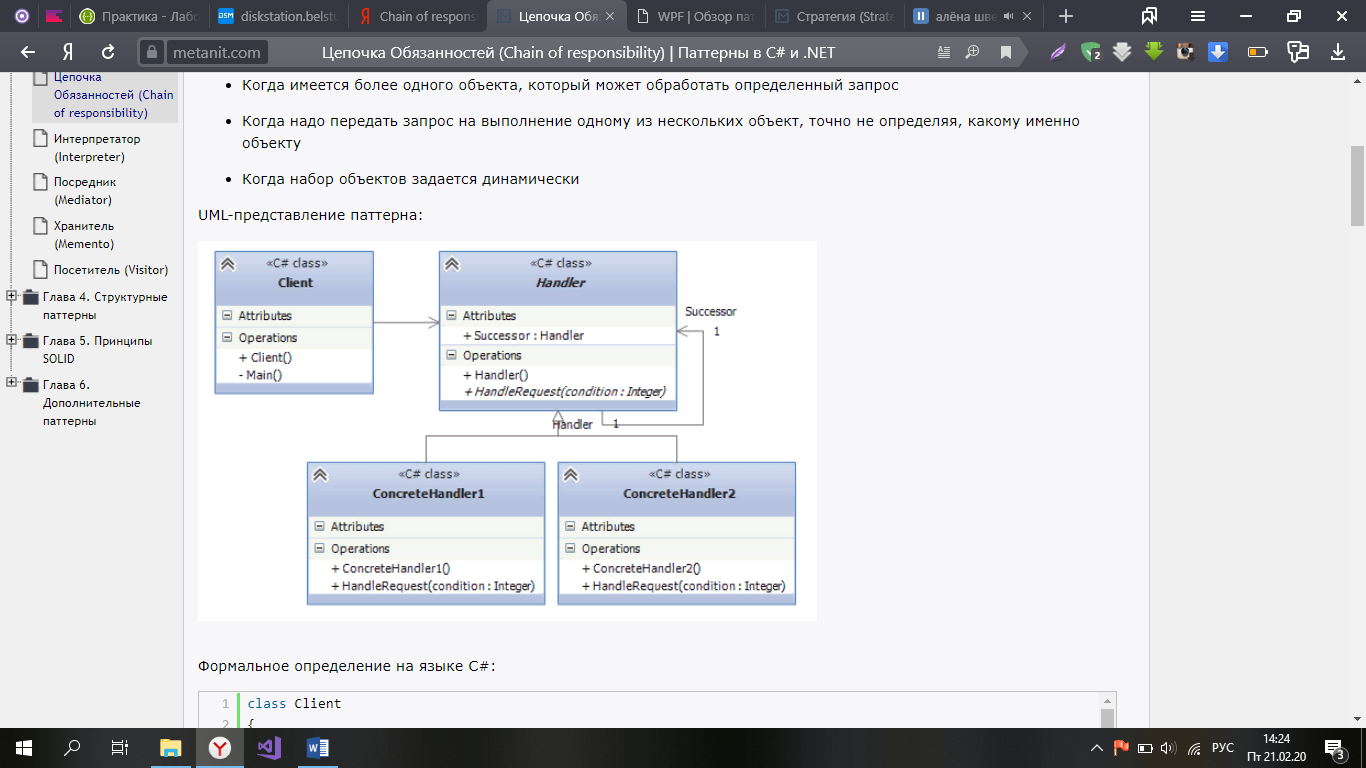
Паттерны поведения были разработаны, чтобы решить проблемы, связанные с поведением и взаимодействием между объектами или классами.

1. **Нарисуете диаграмму классов и поясните принцип работы паттерна Chain of responsibility. В каких случаях надо его применять?**

Цепочка Обязанностей (Chain of responsibility) позволяет избежать жесткой привязки отправителя запроса к получателю, позволяя нескольким объектам обработать запрос. Все возможные обработчики запроса образуют цепочку, а сам запрос перемещается по этой цепочке, пока один из ее объектов не обработает запрос. Каждый объект при получении запроса выбирает, либо обработать запрос, либо передать выполнение запроса следующему по цепочке.

К**огда 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



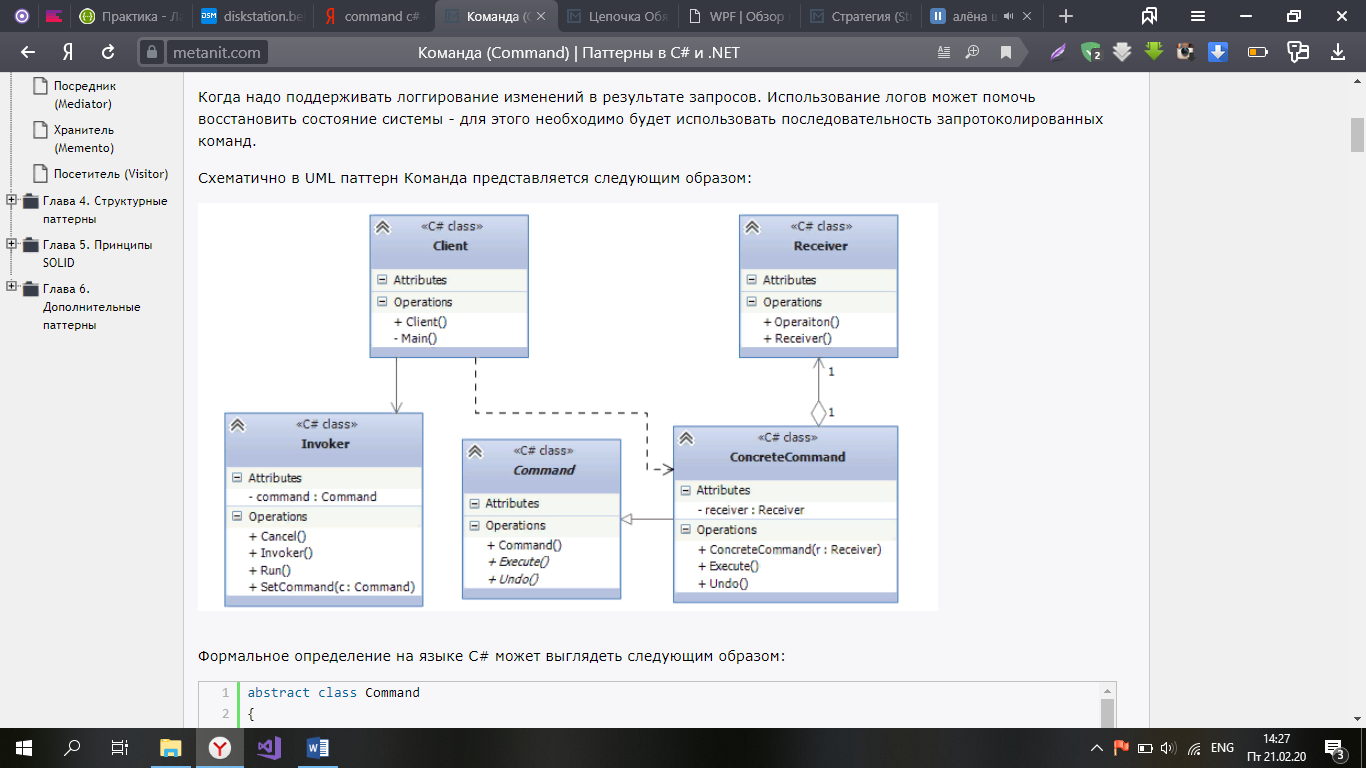
1. **Назначение и принцип организации паттерна Command. Поясните как он связан с конечными автоматами**

Паттерн "Команда" (Command) позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. Этот объект запроса на действие и называется командой. При этом объекты, инициирующие запросы на выполнение действия, отделяются от объектов, которые выполняют это действие.

Команды могут использовать параметры, которые передают ассоциированную с командой информацию. Кроме того, команды могут ставиться в очередь и также могут быть отменены.

**Когда использовать команды?**

* 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.



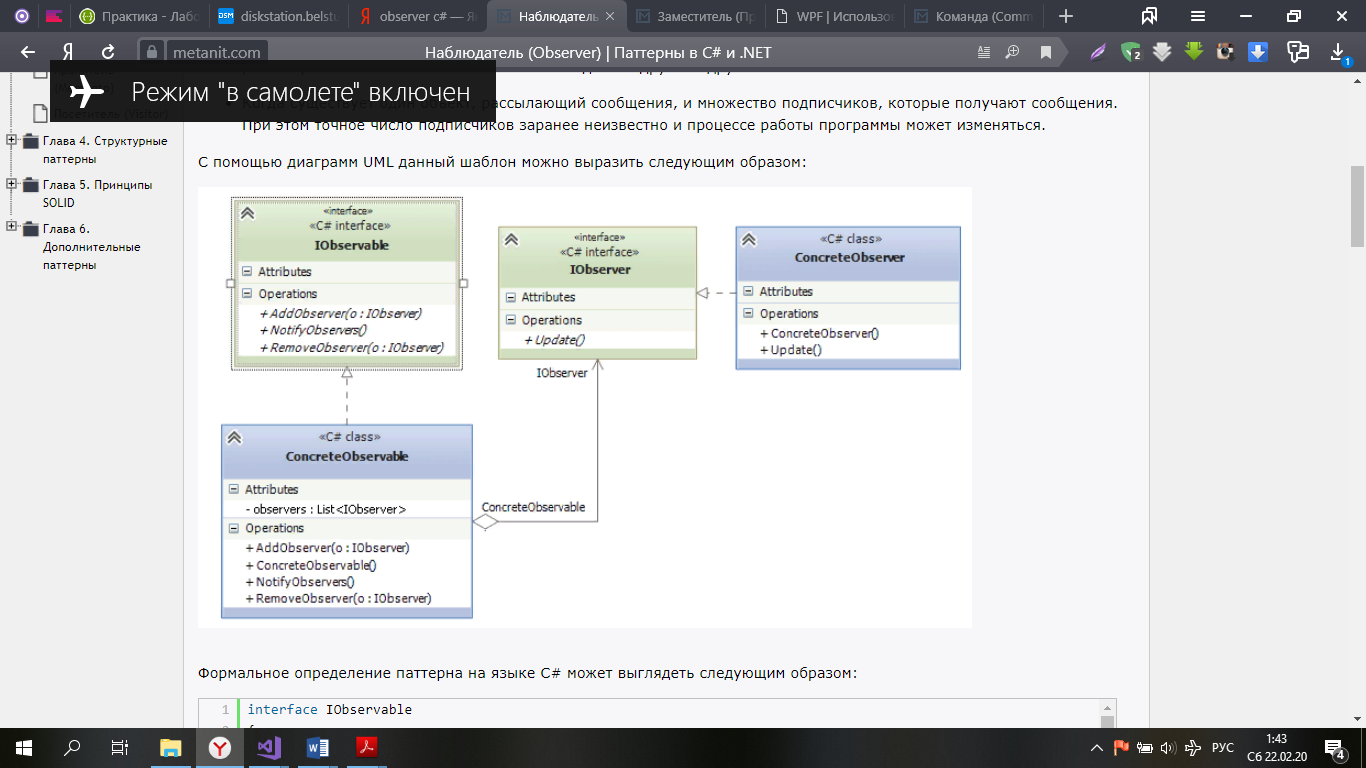
1. **Как реализовать паттерн Observer?**

Паттерн "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.

Данный паттерн еще называют Publisher-Subscriber (издатель-подписчик), поскольку отношения издателя и подписчиков характеризуют действие данного паттерна: подписчики подписываются email-рассылку определенного сайта. Сайт-издатель с помощью email-рассылки уведомляет всех подписчиков о изменениях. А подписчики получают изменения и производят определенные действия: могут зайти на сайт, могут проигнорировать уведомления и т.д.

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.

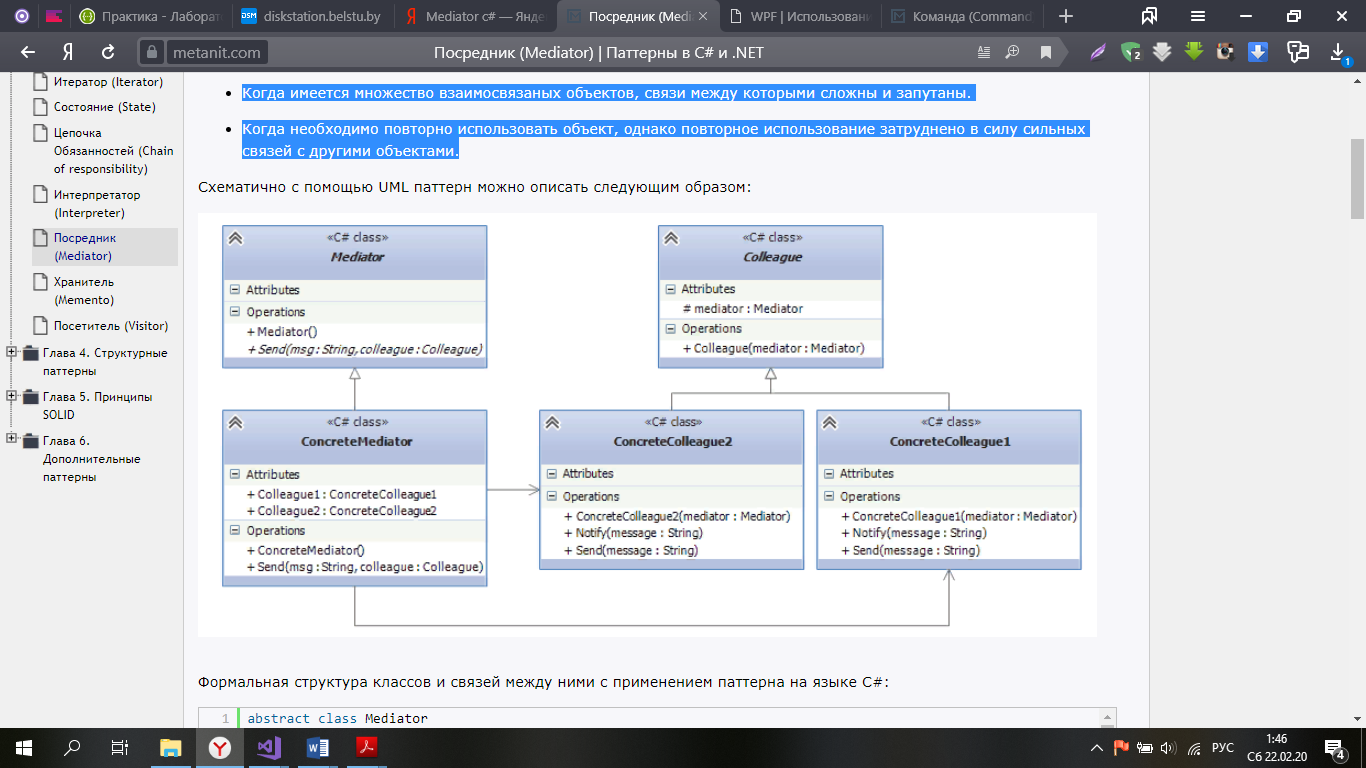


1. **Нарисуйте диаграмму классов для паттерна Mediator. Поясните его назначение.**

Паттерн 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.



1. В чем разница между паттернами Mediator и Facade?

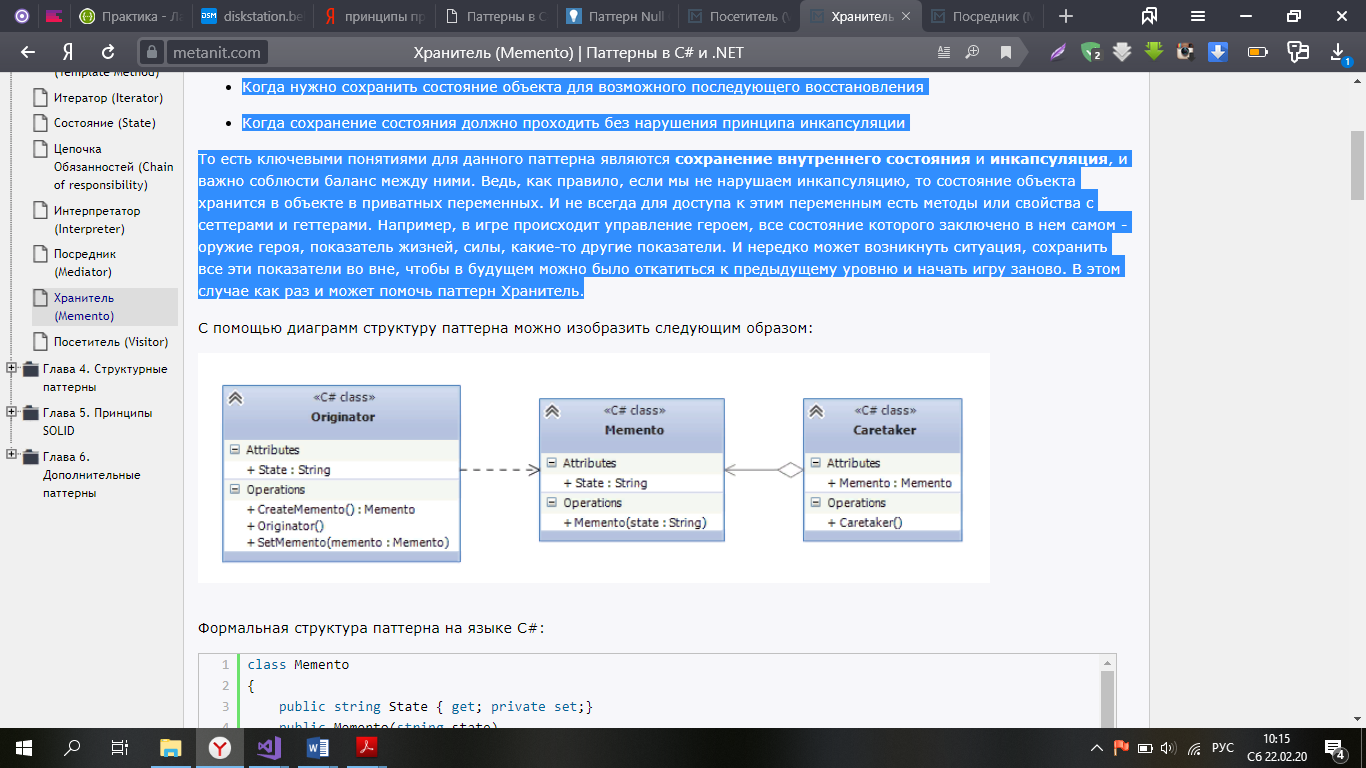
**7. В чем суть паттерна Memento? Поясните на примере.**

Паттерн 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

То есть ключевыми понятиями для данного паттерна являются сохранение внутреннего состояния и инкапсуляция, и важно соблюсти баланс между ними. Ведь, как правило, если мы не нарушаем инкапсуляцию, то состояние объекта хранится в объекте в приватных переменных. И не всегда для доступа к этим переменным есть методы или свойства с сеттерами и геттерами. Например, в игре происходит управление героем, все состояние которого заключено в нем самом - оружие героя, показатель жизней, силы, какие-то другие показатели. И нередко может возникнуть ситуация, сохранить все эти показатели во вне, чтобы в будущем можно было откатиться к предыдущему уровню и начать игру заново. В этом случае как раз и может помочь паттерн Хранитель.



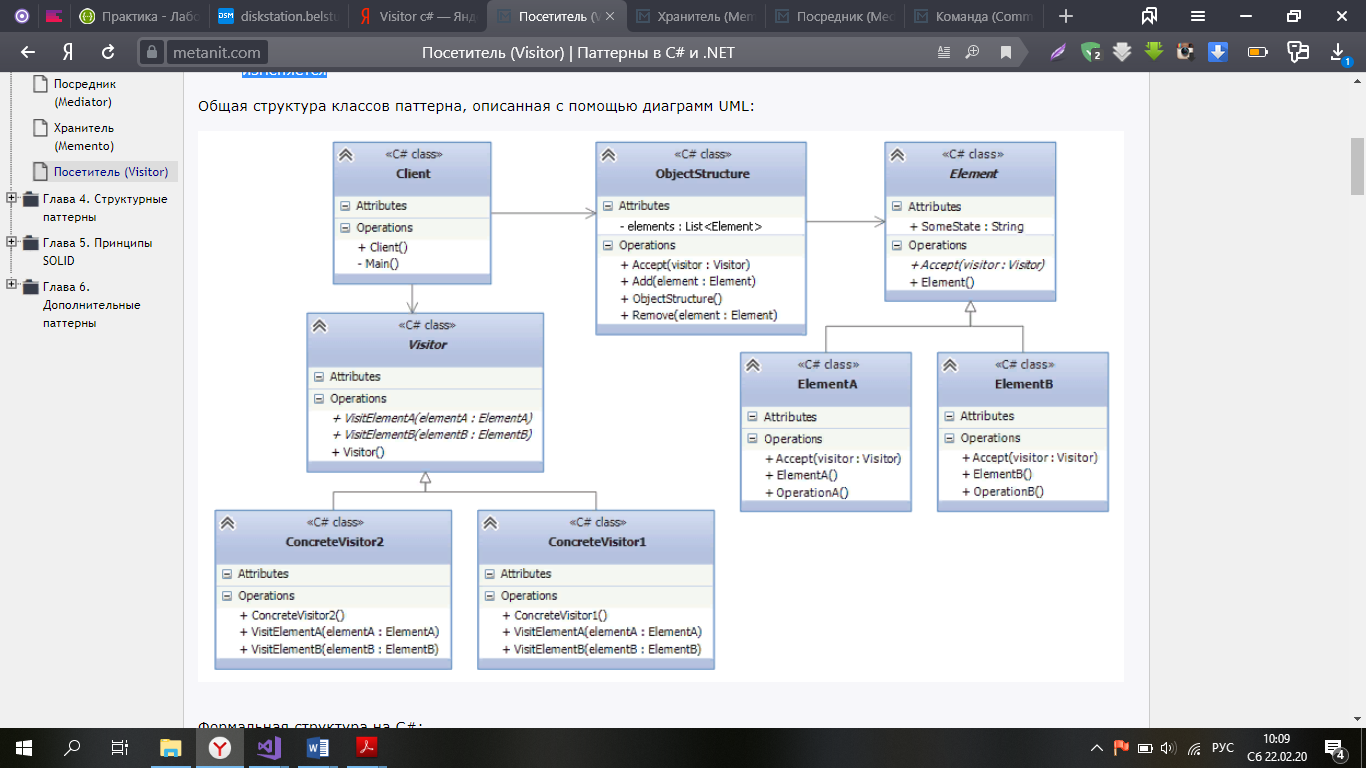
**8. Расскажите о паттерне Visitor?**

Паттерн 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



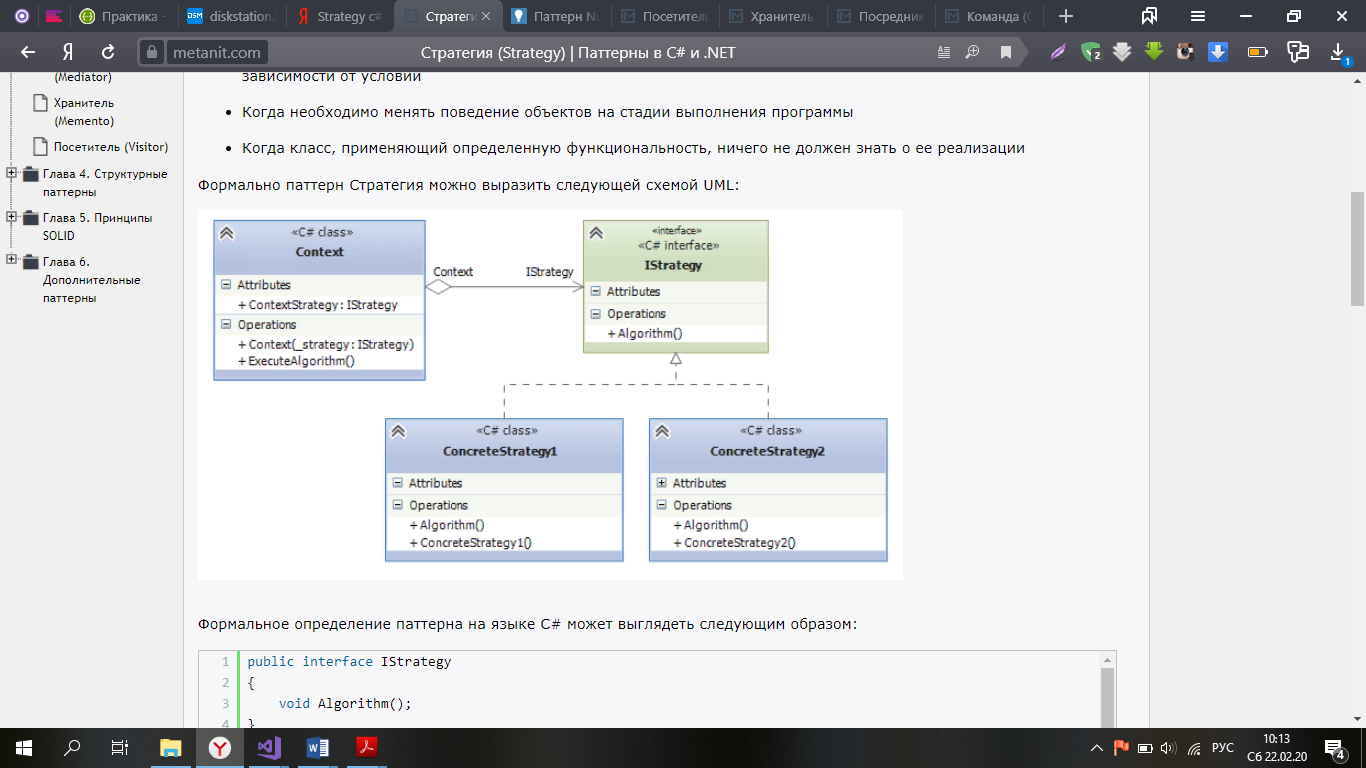
9. В каких случаях надо применять Null object?

**10. Поясните на диаграмме классов как реализовать Strategy.**

Паттерн Стратегия (Strategy) представляет шаблон проектирования, который определяет набор алгоритмов, инкапсулирует 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



11. Перечислите и поясните принципы проектирования SOLID.

***SOLID*** - это 5 принципов объектно-ориентированного программирования, описывающих архитектуру программного обеспечения.



SRP – принцип единой ответственности

Этот принцип означает, что каждый класс или подобная структура в вашем коде должна отвечать только за одну цель. Все члены этого класса должны быть связаны одной целью. Наш класс не должен быть похож на швейцарский нож, в котором при изменении одного из членов нужно изменять весь инструментарий. Это не означает, что ваши классы должны содержать только один метод или свойство. Может быть много членов, если они относятся к единой ответственности.

## OCP - принцип открытости/закрытости

Главной концепцией данного принципа является то, что класс должен быть открыт для расширений, но закрыт от модификаций. Наш модуль должен быть разработан так, чтобы новая функциональность могла быть добавлена только при создании новых требований. «Закрыт для модификации» означает, что мы уже разработали класс, и он прошел модульное тестирование. Мы не должны менять его, пока не найдем ошибки. Как говорится, класс должен быть открытым только для расширений и в C# мы можем использовать для этого наследование.

## Принцип подстановки Лисков (LCP)

Данный принцип гласит, что «вы должны иметь возможность использовать любой производный класс вместо родительского класса и вести себя с ним таким же образом без внесения изменений». Этот принцип прост, но очень важен для понимания. Класс Child не должен нарушать определение типа родительского класса и его поведение

## Принцип разделения интерфейсов (ISP)

Принцип разделения интерфейсов гласит, что клиенты не должны принудительно внедрять интерфейсы, которые они не используют. Что же это означает? Давайте предположим, что есть одна база данных для хранения данных всех типов сотрудников (то есть Junior и Senior). Какой будет лучший подход для нашего интерфейса?

## Принцип инверсии зависимостей (DIP)

Данный принцип гласит, что, во-первых, классы высокого уровня не должны зависеть от низкоуровневых классов. При этом оба должны зависеть от абстракций. Во-вторых, абстракции не должны зависеть от деталей, но детали должны зависеть от абстракций. Классы высокого уровня реализуют бизнес-правила или логику в системе (приложении). Низкоуровневые классы занимаются более подробными операциями, другими словами, они могут заниматься записью информации в базу данных или передачей сообщений в операционную систему или службы и т.п.