**Вопросы к лабораторной работе №19-20**

1. Назначение структурных паттернов?

Структурные паттерны помогают решать задачи с тем, как совмещать и сочетать сущности вместе. Они заботятся о том, как сущности могут использовать друг друга. Простыми словами — они отвечают на вопрос «Как составить программный компонент так, чтобы его можно было компоновать с другими?».

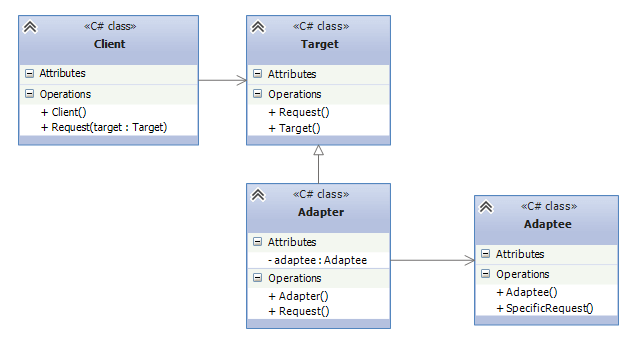
2. Нарисуете диаграмму классов и поясните принцип работы паттерна Adapter. В каких случаях надо его применять.

Паттерн Адаптер (Adapter) предназначен для преобразования интерфейса одного класса в интерфейс другого. Благодаря реализации данного паттерна мы можем использовать вместе классы с несовместимыми интерфейсами.

**Когда надо использовать Адаптер?**

* 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

Формальное определение паттерна на UML выглядит следующим образом:



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

Основное различие:

* **Адаптер класса** использует наследование и может переносить только **класс**. Он не может обернуть интерфейс, поскольку по определению он должен быть производным от некоторого базового класса.
* **Объектный адаптер** использует композицию и может переносить классы или интерфейсы, или и то, и другое. Это можно сделать, поскольку он содержит в качестве частного инкапсулированного элемента **экземпляр объекта** класса или интерфейса, который он переносит.

4. Назначение и принцип организации паттерна Декоратор.

Декоратор (Decorator) представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность.

Для определения нового функционала в классах нередко используется наследование. Декораторы же предоставляет наследованию более гибкую альтернативу, поскольку позволяют динамически в процессе выполнения определять новые возможности у объектов.

### Когда следует использовать декораторы?

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

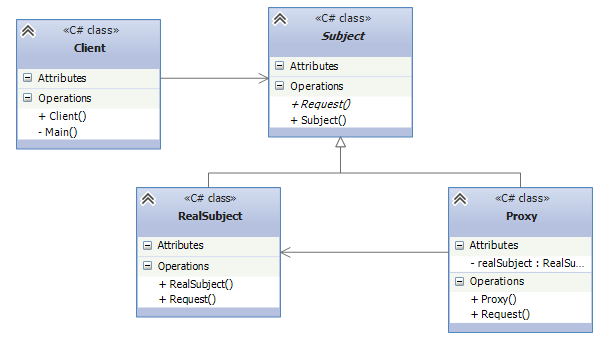
5. Нарисуйте диаграмму классов для паттерна Proxy. Поясните его назначение. Какие разновидности proxy вы знаете?

Паттерн 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)**

С помощью UML паттерн может быть описан так:



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). Прокси - противоположность первых двух, т.к. ее задача - быть прозрачной на стыке взаимодействия типов и не менять интерфейс.

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

Паттерн 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.

Образно 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 Facade?

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

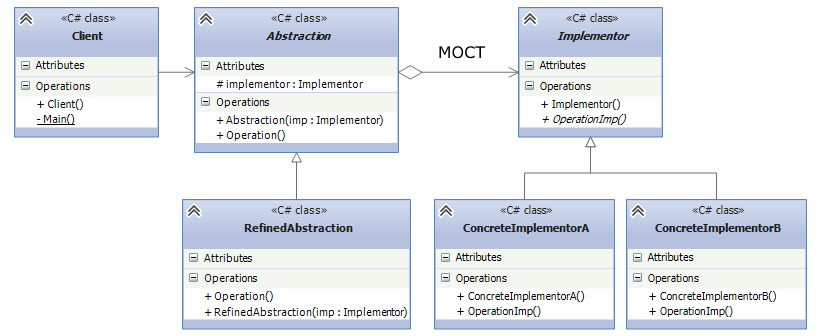
Мост (Bridge) - структурный шаблон проектирования, который позволяет отделить абстракцию от реализации таким образом, чтобы и абстракцию, и реализацию можно было изменять независимо друг от друга.

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

Общая реализация паттерна состоит в объявлении классов абстракций и классов реализаций в отдельных параллельных иерархиях классов.



10.Назначение паттернов поведения?

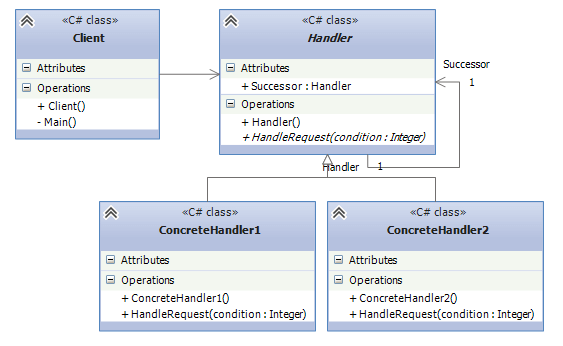
Паттерны поведения связаны с алгоритмами и распределением обязанностей между объектами. Речь в них идет не только о самих объектах и классах, но и о типичных способах взаимодействия. Паттерны поведения характеризуют сложный поток управления, который трудно проследить во время выполнения программы. Внимание акцентировано не на потоке управления как таковом, а на связях между объектами.

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

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.

13.Как реализовать паттерн Observer?

**Наблюдатель** — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

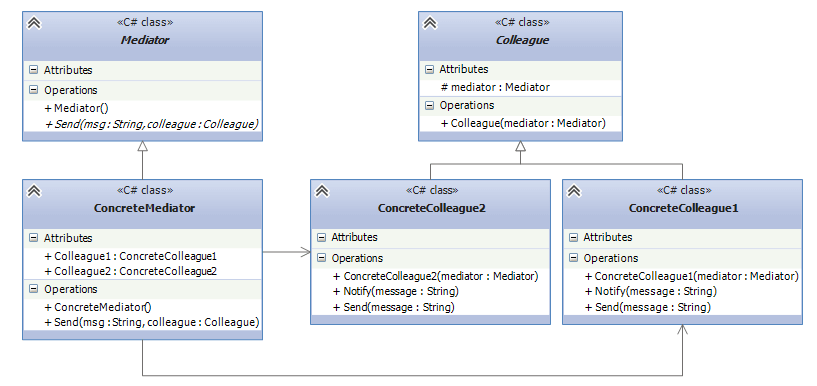
Паттерн Наблюдатель предлагает хранить внутри объекта издателя список ссылок на объекты подписчиков, причём издатель не должен вести список подписки самостоятельно. Он предоставит методы, с помощью которых подписчики могли бы добавлять или убирать себя из списка.

14.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.



15.В чем разница между паттернами Mediator и Facade?

Принципиальное различие между этими двумя паттернами заключается в том, что фасад, как структурный паттерн, всего лишь передает существующую функциональность в медиатор, в то время как медиатор, как поведенческий паттерн, может эту функциональность расширять.

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

Хранитель» (Memento), также известный как Снимок – поведенческий паттерн проектирования. Он позволяет определять, сохранять, а также восстанавливать предыдущие состояния объектов без нарушения принципа инкапсуляции.

17.Расскажите о паттерне Visitor?

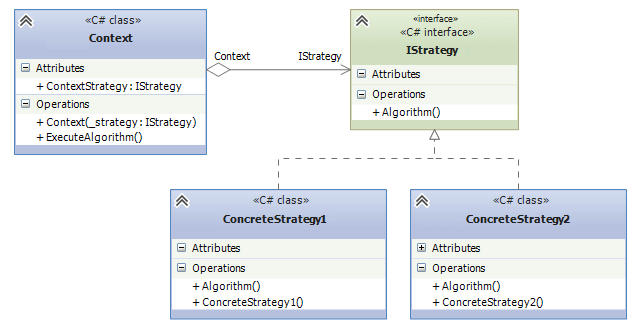
Паттерн Visitor – позволяет единообразно обойти набор элементов с разнородными интерфейсами (т.е. набор объектов разных классов не приводя их к общему базовому типу), а также позволяет добавить новый метод (функцию) в класс объекта, при этом не изменяя сам класс этого объекта.

18.В каких случаях надо применять Null object?

Целью Null-object'а является инкапсулирование отсутствия объекта путём замещения его другим объектом, который ничего не делает.Данный шаблон проектирования рекомендуется использовать, когда:Объект требует взаимодействия с другими объектами. Null Object не устанавливает нового взаимодействия — он использует уже установленное взаимодействие.Какие-то из взаимодействующих объектов должны бездействовать.Требуется абстрагирование «общения» с объектами, имеющими NULL-значение.

19.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.



20.Перечислите и поясните принципы проектирования SOLID.

Вот как расшифровывается акроним SOLID:

* S: Single Responsibility Principle (Принцип единственной ответственности).
* O: Open-Closed Principle (Принцип открытости-закрытости).
* L: Liskov Substitution Principle (Принцип подстановки Барбары Лисков).
* I: Interface Segregation Principle (Принцип разделения интерфейса).
* D: Dependency Inversion Principle (Принцип инверсии зависимостей).

## Принцип единственной ответственности *–*

## *Каждый класс должен решать лишь одну задачу.* Принцип открытости-закрытости *–*

## *Программные сущности (классы, модули, функции) должны быть открыты для расширения, но не для модификации.*

## Принцип подстановки Барбары Лисков *–*

## *Необходимо, чтобы подклассы могли бы служить заменой для своих суперклассов.*

## Принцип разделения интерфейса *–*

## *Создавайте узкоспециализированные интерфейсы, предназначенные для конкретного клиента. Клиенты не должны зависеть от интерфейсов, которые они не используют.*

## Принцип инверсии зависимостей *–*

## *Объектом зависимости должна быть абстракция, а не что-то конкретное.*