**Паттерное проектирование**

***Design patterns*** – это повторяемые архитектурные конструкции, представляющие решение проблемы проектирования в рамках определенного часто возникающего контекста.

***Pattern*** – это не конкретный код, а концепция типичного способа решения той или иной проблемы. Паттерное нужно подстраивать под требование программы.

**1994 -** выпуск книги «Приемы объектно-ориентированного проектирования».

**GoF** – Gang of Four.

Head First. Паттерны проектирования

refactoring.guru

metanit.com c#

Элементы описания паттерна:

1. Проблема, которая решает паттерн
2. Мотивация решения проблем способом, который предлагает паттерн
3. Структура классов, составляющих решение
4. Пример на одном из языков программирования
5. Особенности реализации различных контекстов
6. Связи с другими паттернами

Причины использования паттернов:

1. Проверенные решения
2. Стандартизация кода (типовые решения уменьшают количество ошибок)
3. Упрощение коммуникации

Типы паттерна:

1. **creational** (Порождающий). Обеспечивает гибкое создание объекта без внесения в программу лишних зависимостей, абстрагирует процесс создания проекта, позволяет сделать систему независимой от способа создания композиции и представления объекта. Шаблон, порождающий классы, использует наследование для изменения класса. Шаблоны, порождающие объекты, делегируют создание другим объектам.

**Фабричный метод** (Factory Method) определяет общий интерфейс для создания объектов в родительском классе. Позволяет дочерним классам изменять тип создаваемых объектов.

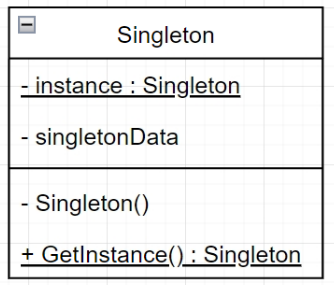
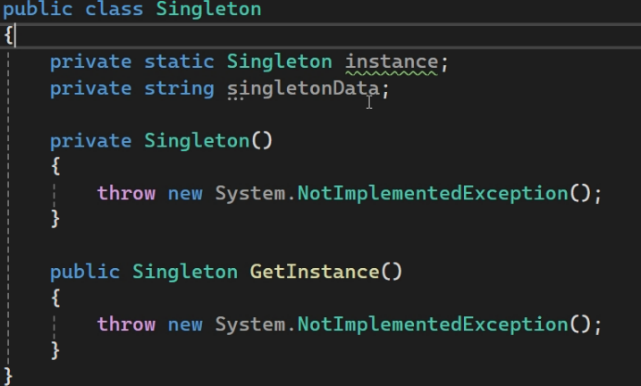
**Абстрактная фабрика** (Abstract Factory) Позваляет создать семейство связанных объектов, не привязываясь к конкретному классу

**Строитель** (Builder). Позволяет пошагово создать сложные объекты и дает возможность получения разных представлений объекта, используя один и тот же код строительство.

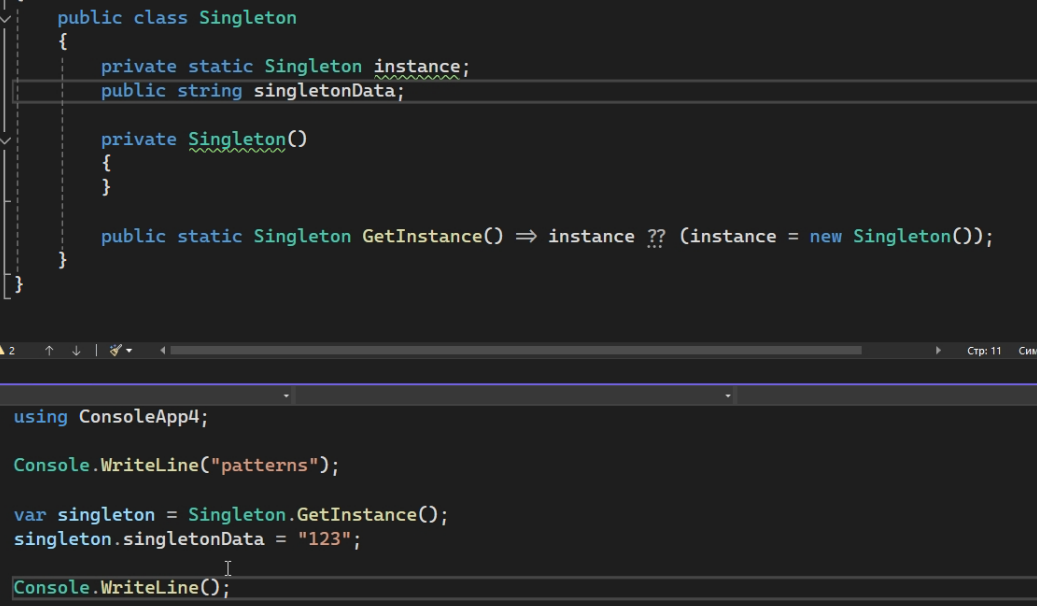
**Прототип**(Prototype**)**. Позволяет копировать объекты, не вдаваясь в подробности их реализации.

Одиночка (Singleton, Синглтон). Гарантирует, что у класса есть только один экземпляр, предоставляет к нему глобальную точку доcтупа.

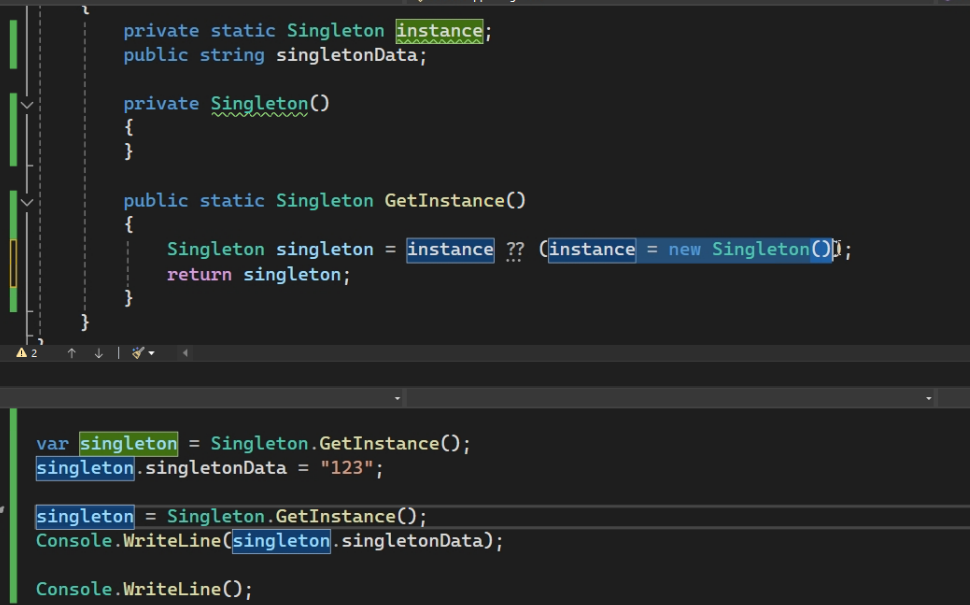
Пример реализации Одиночки. Иногда называют антипаттерном и не рекомендуют к использованию.



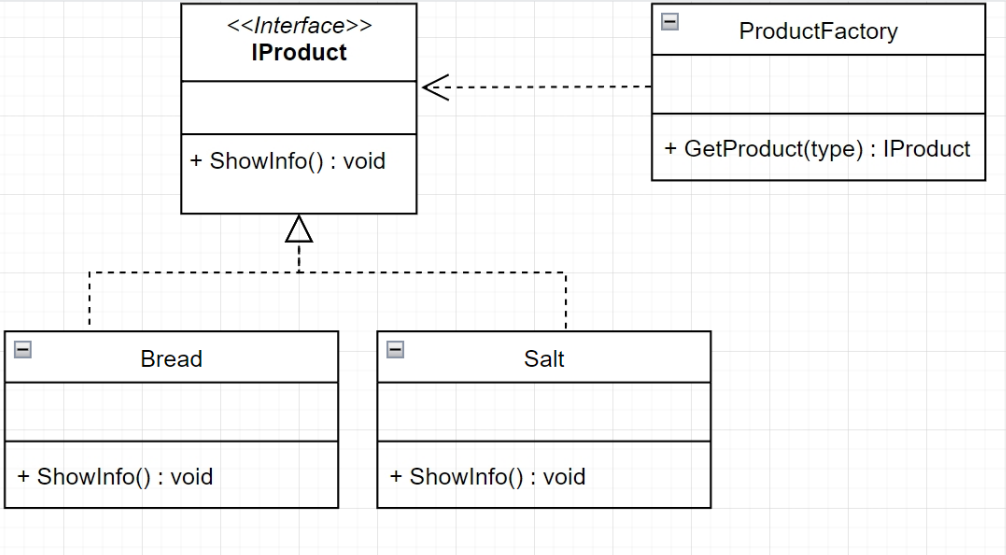
1. **behavioral**
2. **structural**
3. **fundamental**





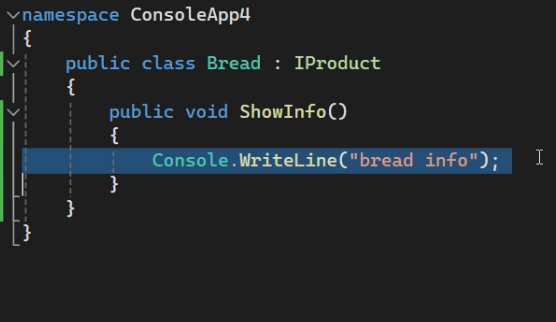


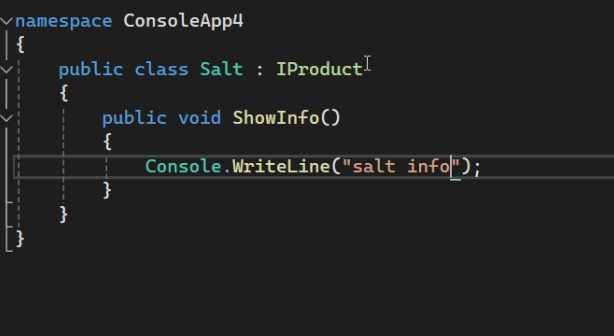
Фабричный метод



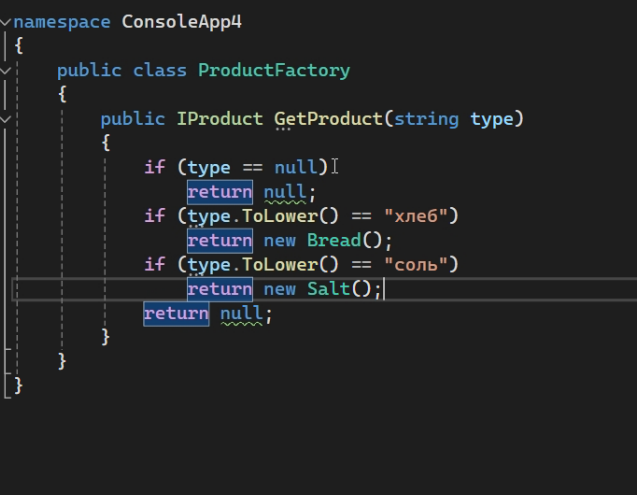
Продолжение лекции (15.03.24)

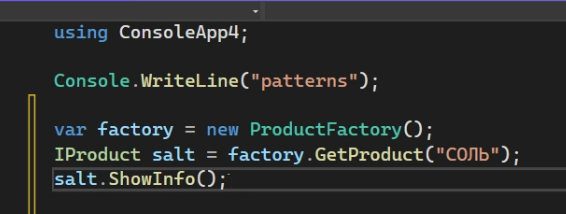
Два дочерних класса (Salt, Bread), реализующие интерфейс IProduct

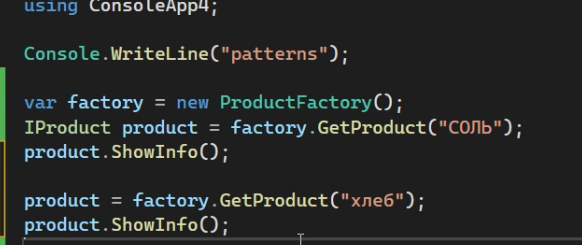




Родительский класс ProductFacroty и его реализация







Поведенческий паттерн:

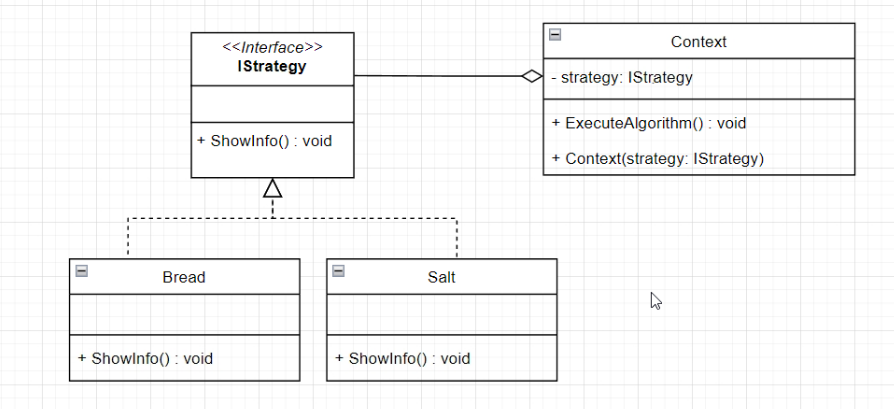
Обеспечивает продуктивную коммуникацию между объектами и определяет взаимодействие между объектами.

1. Стратегия (Strategy)

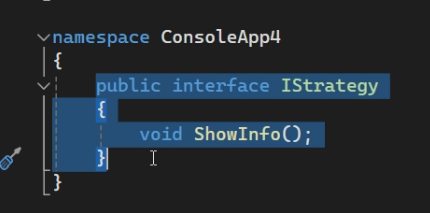
Определяет семейство сложных алгоритмов. Помещает каждый алгоритм в отдельный класс – из-за этого алгоритмы можно заменять во время исполнения программы.

Context – класс.

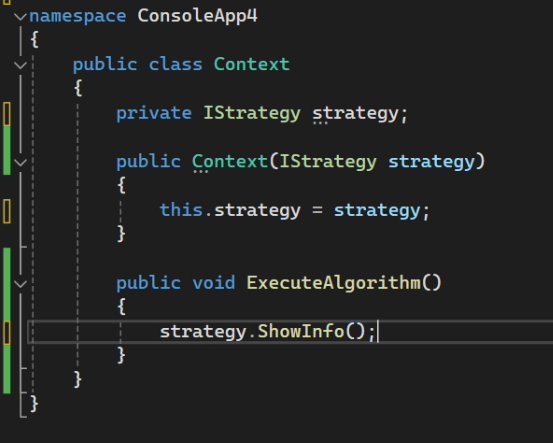
Context (strategy : IStrategy) – конструктор с параметрами (в примере).



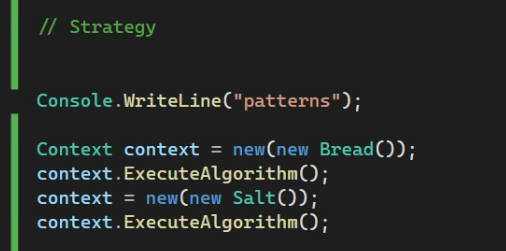
Реализация интерфейса IStrategy:



Реализация класса Context:



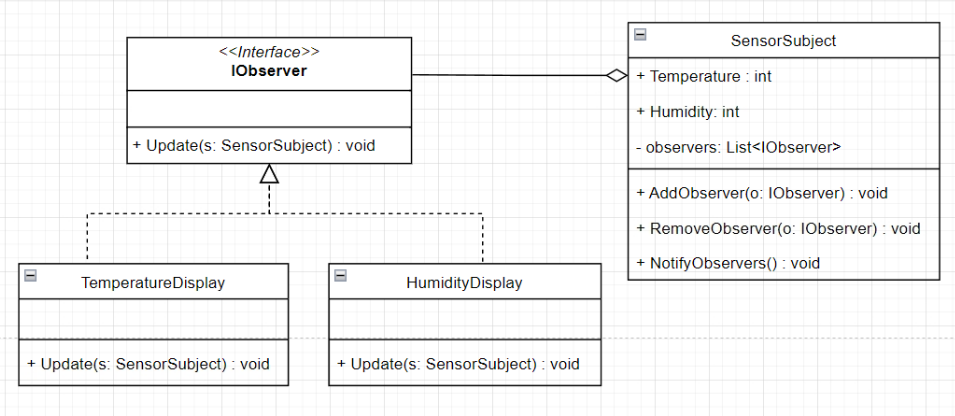
Тестируем в MainProgram:



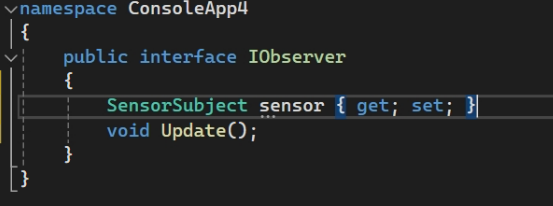
1. Наблюдатель (Observer)

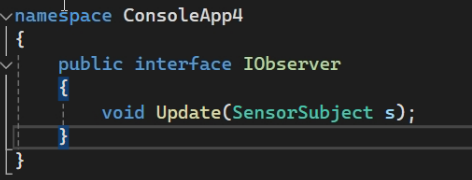
Создает механизм отписки, позволяющий одним объектам следить за изменением состояния другого объекта и реагировать на изменения.

Определяет зависимость 1: М (Пример: подписаться на кого-то в соц. сетях.)

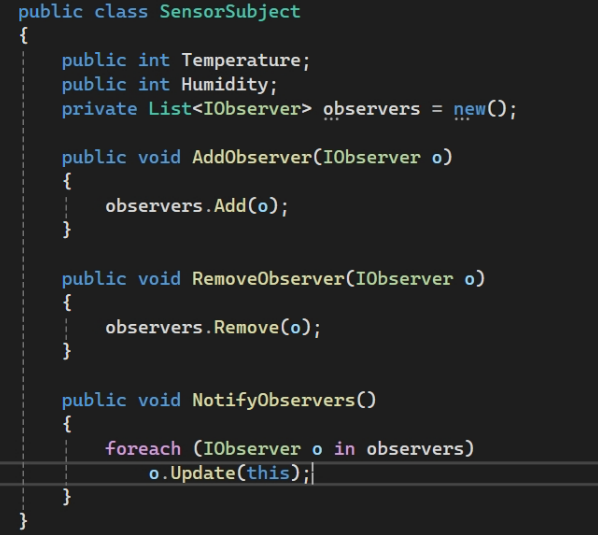


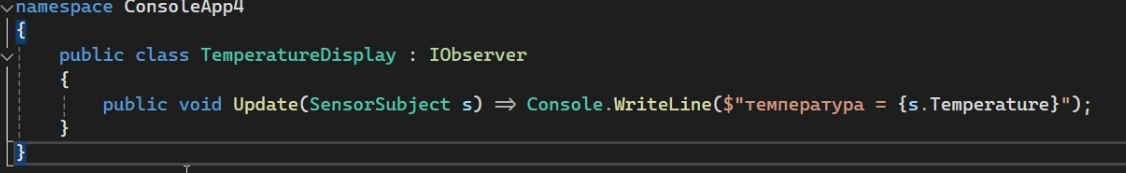
В классе SensorSubject Temperature, Humidity, observers - поля

Реализация интерфейса IObserver:

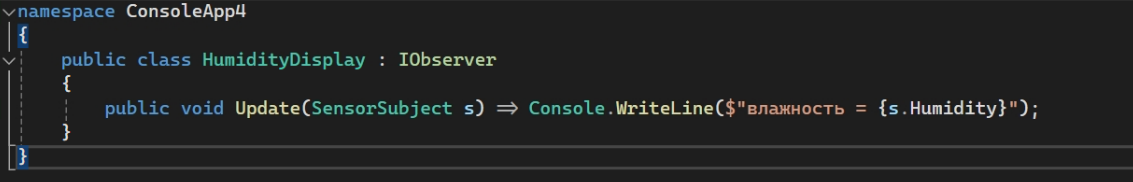


Реализация класса SensorSubject:



Реализация класса TemperatureDisplay:

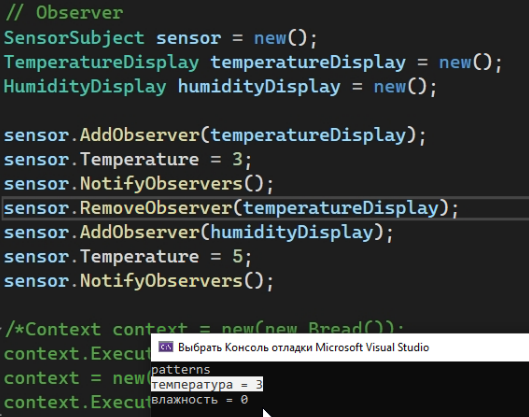
Реализация класса HumiditiDisplsay:



sensor – изменяет данные

temperatureDisplay, humiditiDisplsay – наблюдатели

Тестируем в MainProgram:

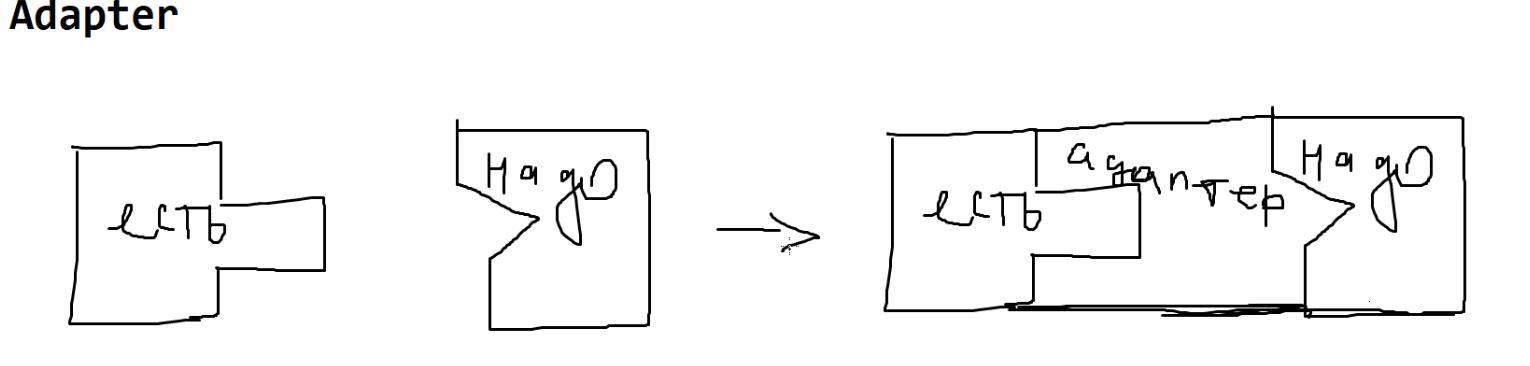


Этапы улучшения:

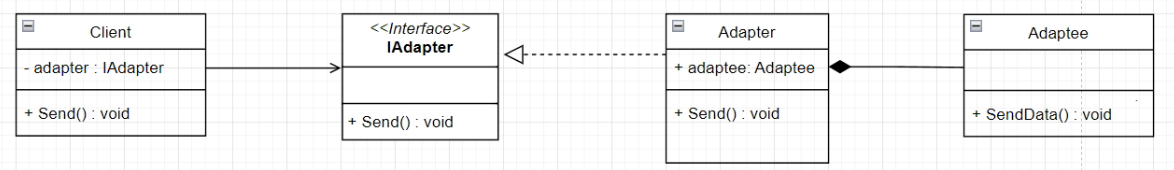
1. В классе Subject вызвать NotifyObervers при изменении значений
2. Вместо передачи Subject параsметры Update добавить в интерфейс свойства типа Subject

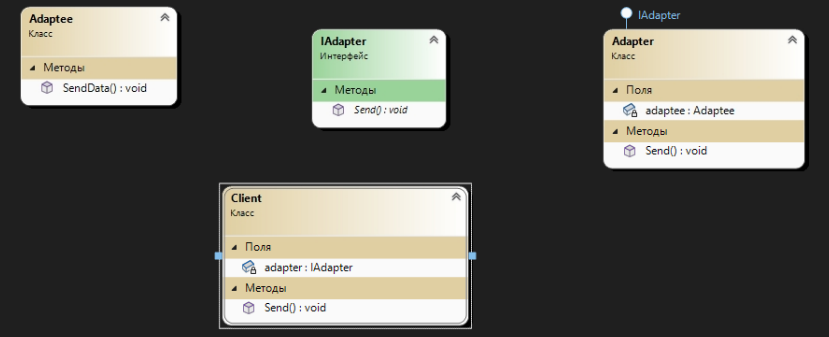
2. Структурные паттерны

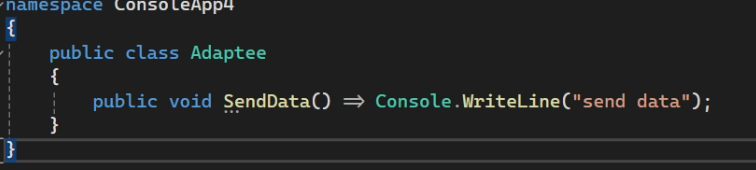
Показывают различные способы построения связи между объектами, определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

1. Адаптер (Adapter)

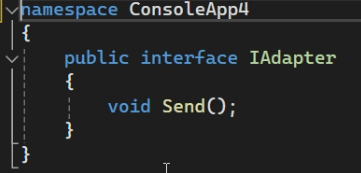
Схемка:



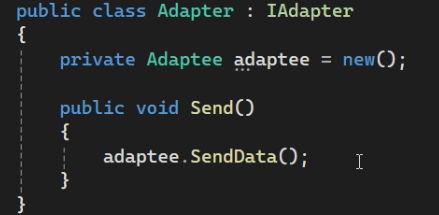


Реализация класса Adaptee:

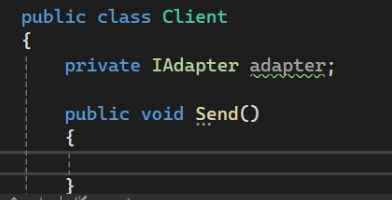
Реализация интерфейса IAdapter:

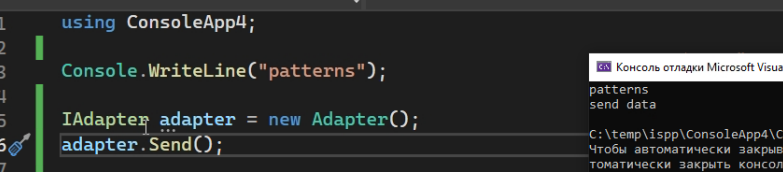


Реализация класса Adapter:



Реализация класса Client:



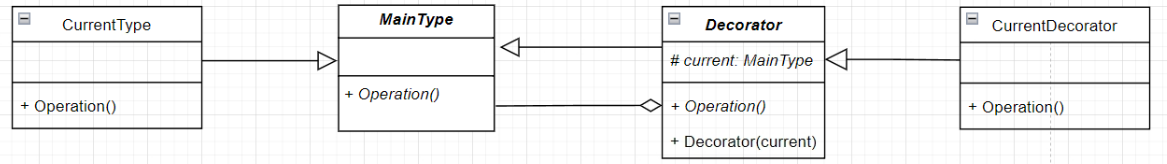
Тестируем в MainProgram:

1. Декоратор (Decorator)

Позволяет динамически добавлять объекту новую функциональность, оборачивая объекты в полезные «обертки».

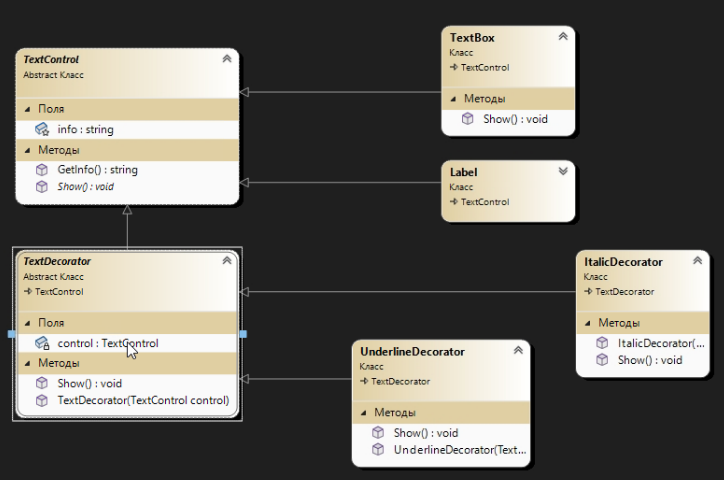
**MainType**, **Decorator** – абстрактные классы.

#*current: MainType –* абстрактный что-то.

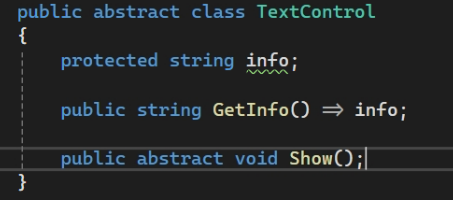


У MainType, Decorator может быть много наследников.

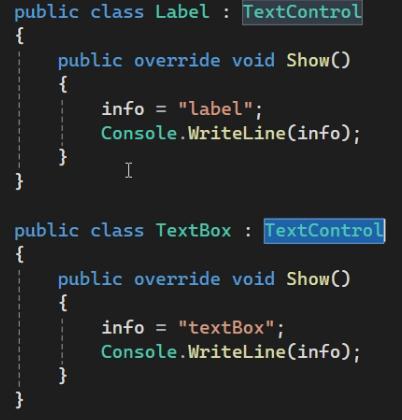
Пример создания элемента управления, отображающего текст с разными декоратами:



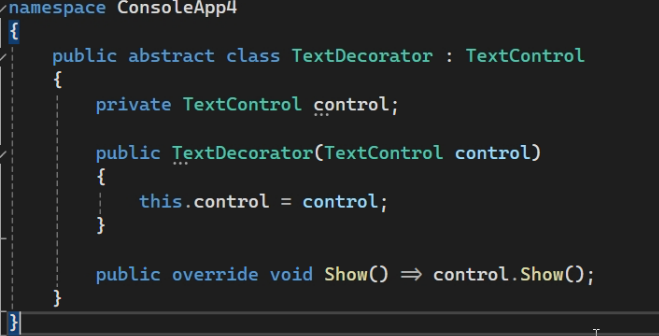
Реализация абстрактного класса TextControl:



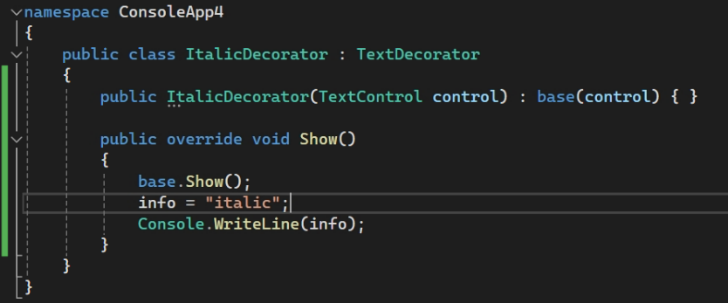
Реализация дочерних классов Label, TextBox:

****

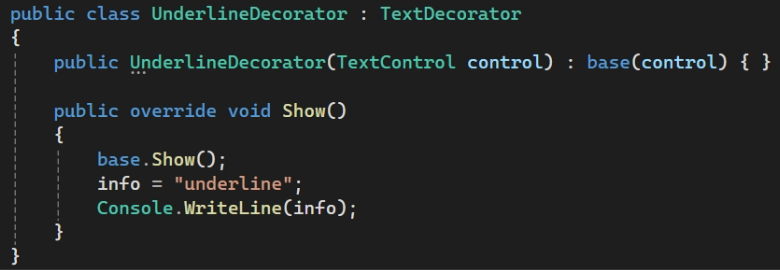
Реализация абстрактного класса TextDecorator:



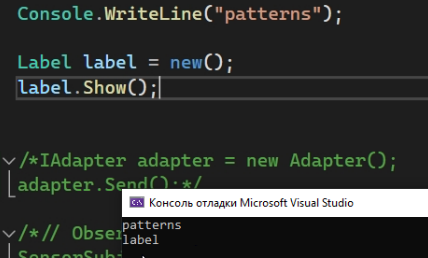
Реализация класса ItalicDecorator:

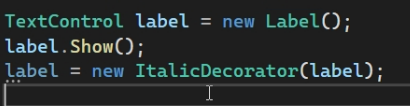


Реализация класса UnderlineDecorator:

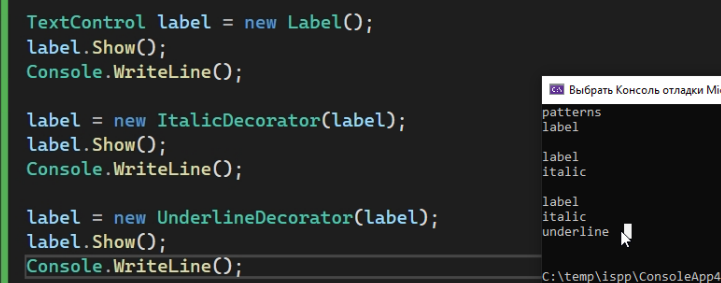


Тестируем в MainProgram:





label = new ItalicDecorator (пишем сюда сам элемент);



3. Мост (Bridge)

Делит классы на несколько классов.

4. Заместитель (Proxy)

Вместо реальных объектов создаются объекты заместители

5. Фасад (Facade)

Упрощает взаимодействие с методами.

6. Компоновщик (Composite)

Позволяет сгруппировать много-много классов в иерархическом виде.

7.Приспособленец (Flyweight)

Нужен для оптимизации хранения объектов памяти.

Поведенческий паттерн:

1. Команда (Command)

Превращает запросы в объекты.

2. Цепочка обязанностей (Chain of responsibility)

Передает вызовы по цепочке.

3. Итератор(Iterator)

Перебор объектов разных типов (думает, что работает с коллекцией).

4. Посредник (Mediator)

Позволяет сделать два класса не такими зависящими друг от друга.

5. Хранитель (Memento)

Сохраняет информацию о том каким объект был ранее.

6. Состояние (State)

Позволяет объекту менять поведение в зависимости от состояния

7. Шаблонный метод (Template Method)

Перекладывает реализацию алгоритма в подклассы.

8. Посетитель (Visitor)

Позволяет добавить новые операции, не изменяя классы, которые эти операции будут использовать.

9. Интерпретатор (Interpreter)

Используется для повторяющихся операции, описания грамматик.