Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО - КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и телекоммуникаций

Кафедра инфокоммуникаций

Реферат

Наблюдатель: Реализация паттерна Observer для обработки событий и уведомлений в Python.

Подготовил студент группы ИВТ-б-о-21-1

_____Костукайло К.Н.

Направление подготовки 09.03.01 Информатика и вычислительная техника

Профиль Автоматизированные системы обработки информации и управления

Проверил

Доцент, кандидат технических наук,

_____ Воронкин Р.А.

Оценка _____

Ставрополь 2024 г.

Наблюдатель: Реализация паттерна Observer для обработки событий и уведомлений в Python

Оглавление

1. Введение.

Определение паттерна Observer.

Паттерн Observer, также известный как паттерн "Издатель-подписчик" (англ. Publisher-Subscriber), представляет собой поведенческий паттерн проектирования, который используется для установления отношения зависимости "один ко многим" между объектами. Основной целью паттерна Observer является определение механизма подписки и оповещения, позволяющего объектам-наблюдателям автоматически реагировать на изменения, происходящие в объектах-субъектах.

Субъект (или издатель) представляет объект, чье состояние может изменяться, и содержит список ссылок на своих наблюдателей. Наблюдатель (или подписчик) представляет объект, желающий получать уведомления об изменениях в субъекте и реагировать на эти изменения соответствующим образом. При изменении состояния субъекта все его наблюдатели автоматически уведомляются о произошедших изменениях и обновляются в соответствии с новым состоянием субъекта.

Применение паттерна Observer способствует созданию слабосвязанных систем, где субъект и наблюдатели могут взаимодействовать без явных зависимостей друг от друга. Это обеспечивает гибкость, расширяемость и переиспользуемость кода, поскольку новые типы наблюдателей могут быть добавлены без изменения субъекта, а изменения в субъекте могут автоматически приводить к обновлению всех его наблюдателей.

В различных областях программирования, включая пользовательские интерфейсы, системы управления базами данных, финансовые приложения и другие, паттерн Observer находит широкое применение для обработки событий и уведомлений, обеспечивая эффективное управление изменениями и обновлениями в приложениях.

Значение реализации Observer для обработки событий и уведомлений

Реализация паттерна Observer для обработки событий и уведомлений представляет собой эффективный механизм, позволяющий разработчикам создавать гибкие, расширяемые и модульные системы, способные автоматически реагировать на изменения состояния объектов и генерацию событий.

Значение данной реализации заключается в следующем:

1. **Отделение обязанностей:** Паттерн Observer помогает отделять логику обработки событий от объектов, которые их генерируют, а также от объектов, которые должны на них отреагировать. Это позволяет создавать чистый и понятный код, где каждый компонент системы выполняет свою специфическую функцию без излишних зависимостей.
2. **Гибкость и расширяемость:** Реализация Observer позволяет добавлять новые типы событий и наблюдателей без изменения существующего кода. Это обеспечивает гибкость системы, позволяя ей эволюционировать и адаптироваться к новым требованиям и условиям.
3. **Уведомление и обновление объектов:** Система, основанная на паттерне Observer, обеспечивает автоматическое уведомление всех заинтересованных наблюдателей при возникновении событий или изменении состояния объекта. Это позволяет подписчикам (наблюдателям) быть в курсе происходящих изменений и проводить соответствующие обновления в ответ на эти изменения.
4. **Снижение связанности:** Использование паттерна Observer снижает связанность между компонентами системы, поскольку объекты-субъекты и объекты-наблюдатели взаимодействуют друг с другом через абстрактный интерфейс, не зная конкретных деталей друг о друге. Это способствует повышению переиспользуемости и понижению уровня зависимостей между компонентами системы.

Таким образом, реализация паттерна Observer представляет собой мощный инструмент для обработки событий и уведомлений, обеспечивая эффективное и гибкое управление потоком информации и процессами в программных системах.

2. Основные принципы паттерна Observer

Объяснение основных концепций паттерна Observer

Основные концепции паттерна Observer базируются на установлении отношения зависимости "один ко многим" между объектами. В контексте данного паттерна, субъект (или издатель) представляет объект, чье состояние может изменяться, а наблюдатель (или подписчик) - объект, желающий получать уведомления об изменениях в субъекте.

Рассмотрим следующие абстрактные классы для иллюстрации концепций паттерна Observer:

```
from abc import ABC, abstractmethod

# Абстрактный класс Субъекта
class Subject(ABC):
    @abstractmethod
    def attach(self, observer):
        pass

    @abstractmethod
    def detach(self, observer):
        pass

    @abstractmethod
    def notify(self):
        pass


# Абстрактный класс Наблюдателя
class Observer(ABC):
    @abstractmethod
    def update(self):
        pass
```

В данном примере класс Subject представляет субъекта, который имеет методы для добавления, удаления и уведомления наблюдателей. Класс

Observer - абстрактный класс, определяющий метод update, который будет вызываться при уведомлении наблюдателей.

Пример конкретной реализации с использованием этих абстрактных классов может выглядеть следующим образом:

```
# Конкретный класс Субъекта
class ConcreteSubject(Subject):
    def __init__(self):
        self._observers = []

    def attach(self, observer):
        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer):
        if observer in self._observers:
            self._observers.remove(observer)

    def notify(self):
        for observer in self._observers:
            observer.update()


# Конкретный класс Наблюдателя
class ConcreteObserver(Observer):
    def update(self):
        print("Получено уведомление об изменениях")
```

В данном примере класс ConcreteSubject предоставляет конкретную реализацию субъекта, а класс ConcreteObserver - конкретную реализацию наблюдателя.

Центральными концепциями паттерна Observer являются субъект (издатель) и наблюдатель (подписчик). Субъект представляет объект, чье состояние может изменяться, и содержит список ссылок на своих наблюдателей. Наблюдатель, в свою очередь, представляет объект, желающий получать уведомления об изменениях в субъекте и реагировать на эти изменения соответствующим образом.

При использовании паттерна Observer субъекты и наблюдатели взаимодействуют через абстрактный интерфейс, позволяющий субъектам оповещать наблюдателей о произошедших изменениях и обновлять их

состояние. Это обеспечивает автоматическое уведомление всех заинтересованных наблюдателей при возникновении событий или изменении состояния субъекта, сохраняя при этом слабую связанность между ними.

Ключевым моментом в концепции паттерна Observer является то, что субъект не зависит от конкретных типов наблюдателей, а наблюдатели, в свою очередь, могут быть подписаны на несколько субъектов одновременно. Это обеспечивает гибкость и переиспользуемость кода, позволяя создавать расширяемые системы, способные эффективно реагировать на изменения внутри приложений.

Таким образом, паттерн Observer обеспечивает эффективный механизм управления потоком информации и событий в программных системах, основанный на принципе "издатель-подписчик", что делает его важным инструментом при разработке многопоточных, асинхронных и событийно-ориентированных приложений.

Примеры использования в реальной жизни.

Примеры использования паттерна Observer в реальной жизни могут быть обнаружены в различных областях, таких как программирование пользовательских интерфейсов, управление базами данных, финансовые приложения и другие системы, где требуется эффективное управление событиями и уведомлениями.

Пример: Использование Observer в финансовых приложениях

В финансовых приложениях паттерн Observer может быть использован для отслеживания изменений котировок, ценных бумаг или других финансовых параметров. Наблюдатели могут быть представлены различными компонентами приложения, такими как графики, отчеты или уведомления, которые должны автоматически обновляться при изменении цен или котировок.

Допустим, у нас есть финансовое приложение, отслеживающее изменение цен акций на бирже. Мы хотим, чтобы пользователи могли подписаться на определенные акции и получать уведомления об изменениях цен. В этом случае мы можем использовать паттерн Observer для реализации механизма уведомлений.

```
from abc import ABC, abstractmethod

# Абстрактный класс Субъекта
class Subject(ABC):
    @abstractmethod
    def attach(self, observer):
        pass

    @abstractmethod
    def detach(self, observer):
        pass

    @abstractmethod
    def notify(self):
        pass

# Абстрактный класс Наблюдателя
class Observer(ABC):
    @abstractmethod
    def update(self, symbol, price):
        pass

# Конкретный класс Субъекта - Биржевая информация
class StockMarket(Subject):
    def __init__(self):
        self._observers = []

    def attach(self, observer):
        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer):
        if observer in self._observers:
            self._observers.remove(observer)

    def notify(self, symbol, price):
        for observer in self._observers:
            observer.update(symbol, price)

# Конкретный класс Наблюдателя - Подписчик акций
class StockSubscriber(Observer):
    def __init__(self, name):
        self.name = name

    def update(self, symbol, price):
        print(f"{self.name} received notification: {symbol} price is {price}")

# Создание экземпляров объектов
stock_market = StockMarket()
```

```
subscriber1 = StockSubscriber("Alice")
subscriber2 = StockSubscriber("Bob")

# Подписка на уведомления
stock_market.attach(subscriber1)
stock_market.attach(subscriber2)

# Имитация изменения цены акции и отправка уведомлений
stock_market.notify("AAPL", 150.00)
```

В данном примере класс StockMarket выступает в качестве конкретного субъекта, который отслеживает изменения цен акций. Класс StockSubscriber представляет конкретного наблюдателя, который подписывается на уведомления об изменениях цен.

Этот пример демонстрирует применение паттерна Observer для реализации уведомлений об изменении цен акций в реальном приложении, что делает его важным инструментом для разработки финансовых систем и других приложений, требующих эффективной обработки событий и уведомлений.

Пример: Интерактивные пользовательские интерфейсы

При разработке интерактивных пользовательских интерфейсов паттерн Observer может быть использован для реализации отслеживания изменений состояния объектов. Например, кнопки, чекбоксы или ползунки могут действовать как издатели, а блоки информации или графические элементы - как подписчики. При изменении состояния издателя все подписчики автоматически уведомляются о произошедших изменениях и обновляются соответствующим образом.

Пример: Управление базами данных

В системах управления базами данных паттерн Observer может использоваться для отслеживания изменений в базе данных и уведомления заинтересованных компонентов о произошедших изменениях. Например, при добавлении новой записи в базу данных, все зарегистрированные наблюдатели могут быть автоматически уведомлены о появлении новой информации.

Пример: Мобильные уведомления

В мобильных приложениях паттерн Observer может быть применен для реализации системы уведомлений, где устройства-получатели уведомлений (наблюдатели) получают автоматические уведомления от серверов или других устройств-издателей при наступлении определенных событий, таких как новые сообщения, обновления или предупреждения.

Такие реальные примеры использования паттерна Observer позволяют создавать гибкие и эффективные системы, способные автоматически реагировать на изменения и события внутри приложений, обеспечивая эффективное управление потоком информации и процессами.

3. Реализация паттерна Observer в Python

Использование встроенных инструментов

В языке программирования Python встроенные инструменты для реализации паттерна Observer представлены модулями Observable и Observer из стандартной библиотеки import abc.

Применение встроенных инструментов паттерна Observer в Python начинается с создания абстрактного класса субъекта, который расширяет класс Observable и содержит методы для регистрации, удаления и уведомления наблюдателей. Далее создается конкретный класс субъекта, который наследует абстрактный класс и реализует логику изменения своего состояния.

```
import abc

# Абстрактный класс субъекта - Наблюдаемый объект
class Subject(abc.ABC):
    def __init__(self):
        self._observers = set()

    def register_observer(self, observer):
        self._observers.add(observer)

    def unregister_observer(self, observer):
        self._observers.remove(observer)

    def notify_observers(self, data):
        for observer in self._observers:
```

```
            observer.update(data)

# Конкретный класс субъекта - Издатель
class Publisher(Subject):
    def update_data(self, data):
        self.notify_observers(data)

# Абстрактный класс наблюдателя
class Observer(abc.ABC):
    @abc.abstractmethod
    def update(self, data):
        pass

# Конкретный класс наблюдателя - Подписчик
class Subscriber(Observer):
    def update(self, data):
        print(f"Received data: {data}")
```

В данном примере, класс Publisher служит в качестве конкретного субъекта, способного изменять свое состояние и уведомлять зарегистрированных наблюдателей. Класс Subscriber представляет конкретного наблюдателя, который реагирует на уведомления от субъекта.

Использование встроенных инструментов паттерна Observer в Python обеспечивает возможность разработки гибких и расширяемых систем, где объекты могут взаимодействовать и реагировать на изменения друг друга. Это способствует уменьшению связанности между компонентами системы и облегчает повторное использование кода.

Таким образом, использование встроенных инструментов паттерна Observer в Python представляет собой важный инструмент для построения модульных и поддерживаемых систем, способствуя увеличению гибкости и эффективности разработки программного обеспечения.

Создание пользовательского класса Observer.

Для создания пользовательского класса Observer в Python необходимо определить абстрактный класс наблюдателя, который будет содержать метод обновления (update) для реагирования на изменения состояния субъекта. Далее разработчик может создать конкретные классы наблюдателей, которые будут реализовывать этот метод.

Пример создания пользовательского класса Observer в Python с использованием модуля abc из стандартной библиотеки:

```
import abc

# Абстрактный класс наблюдателя
class Observer(abc.ABC):
    @abc.abstractmethod
    def update(self, data):
        """
        Метод update должен быть переопределен в конкретных классах наблюдателей.

        Args:
        data: Обновленные данные, на которые реагирует наблюдатель.
        """
        pass

# Конкретный класс наблюдателя
class ConcreteObserver(Observer):
    def update(self, data):
        """
        Реализация метода update, позволяющая конкретному наблюдателю реагировать на обновленные данные.

        Args:
        data: Обновленные данные, на которые реагирует наблюдатель.
        """
        print(f"Received updated data: {data}")
```

В данном примере, класс ConcreteObserver представляет конкретного наблюдателя, который переопределяет метод update для реакции на обновленные данные от субъекта.

Создание пользовательского класса Observer позволяет разработчикам определить специфичное поведение для каждого наблюдателя в соответствии с требованиями конкретной системы или приложения, что способствует гибкости и возможности повторного использования кода.

Таким образом, создание пользовательского класса Observer в Python позволяет эффективно реализовывать паттерн Observer и обеспечивает гибкость в разработке систем, способных реагировать на изменения в других компонентах программного обеспечения.

4. Применение Observer для обработки событий и уведомлений

Реальные примеры использования в Python-проектах

1. Графический интерфейс пользователя (GUI) с обновлением

Описание: В графических приложениях, особенно тех, которые работают с динамическими данными, часто используется паттерн Observer для обновления пользовательского интерфейса при изменении данных.

Пример: Представьте, что у вас есть приложение для мониторинга финансовых данных, где цены акций обновляются в режиме реального времени. Вы можете использовать паттерн Observer, чтобы оповестить графический интерфейс о любых изменениях в ценах акций и автоматически обновить отображаемую информацию на экране.

2. Система уведомлений и событий

Описание: Системы уведомлений и событий широко используют паттерн Observer для передачи и обработки сообщений между компонентами системы.

Пример: Веб-приложение, предоставляющее систему уведомлений для пользователей, может использовать паттерн Observer для передачи уведомлений от сервера к клиентскому интерфейсу. Когда происходит новое событие (например, новое сообщение или обновление), зарегистрированные наблюдатели (подписчики) получают уведомление и могут отреагировать соответственно.

3. Мониторинг и журналирование

Описание: Системы мониторинга и журналирования данных могут использовать паттерн Observer для отслеживания изменений в состоянии системы и регистрации этих изменений.

Пример: Представьте, что у вас есть приложение мониторинга производительности серверов. Вы можете использовать паттерн Observer для создания механизма, который будет отслеживать различные метрики (нагрузку процессора, использование памяти, количество запросов и т. д.) и регистрировать их в журнале. Когда любая метрика превышает установленный порог, зарегистрированные наблюдатели могут обработать это событие.

Каждый из этих сценариев демонстрирует применимость паттерна Observer в реальных проектах. Реализация данного паттерна способствует созданию гибких и расширяемых систем, способных реагировать на изменения и обеспечивать взаимодействие между компонентами программного обеспечения.

Как Observer упрощает обработку событий и уведомлений.

1. Паттерн Observer в упрощении обработки событий и уведомлений: Паттерн Observer, также известный как Publish-Subscribe, является ключевым элементом в обработке событий и уведомлений в программировании. Он способствует эффективной организации системы уведомлений, обеспечивая гибкость, расширяемость и минимизацию связанности между компонентами приложения.

2. **Разделение обязанностей:** Паттерн Observer разделяет ответственность между издателем (или субъектом) и наблюдателями (или подписчиками). Издатель ничего не знает о конкретных наблюдателях и может уведомлять любое количество подписчиков без изменения своего кода. Это разделение обязанностей способствует уменьшению связанности и обеспечивает гибкость системы.

3. Уведомление о событиях: Издатель уведомляет своих наблюдателей о возникших событиях или изменениях в состоянии. Наблюдатели реагируют на эти уведомления, выполняя соответствующие действия. Благодаря этому механизму, издатель и наблюдатели остаются

независимыми друг от друга, что способствует легкости добавления новой функциональности без изменения существующего кода.

4. **Расширяемость:** Паттерн Observer обеспечивает высокую степень расширяемости системы. Новые наблюдатели могут быть добавлены без изменения существующего кода издателя, а также новые издатели могут быть созданы, не затрагивая существующих наблюдателей. Это позволяет эффективно адаптировать систему к новым требованиям и сценариям использования.

5. **Множественная рассылка:** Издатель способен уведомлять множество наблюдателей одновременно, что обеспечивает эффективное распространение информации о событиях по всей системе. Это уменьшает издержки и повышает производительность обработки событий.

6. Система событий: Паттерн Observer позволяет создавать систему подписки на события, где издатель активно уведомляет своих подписчиков о произошедших событиях. Это упрощает реализацию архитектуры, в которой различные компоненты приложения могут реагировать на события без явной зависимости друг от друга, что способствует созданию более модульной и отзывчивой системы.

Таким образом паттерн Observer представляет собой мощный инструмент для обработки событий и уведомлений, обеспечивая гибкость, расширяемость и разделение ответственностей между компонентами приложения. Его использование способствует построению отзывчивых и эффективных систем, способных эффективно реагировать на изменения и взаимодействовать между компонентами программного обеспечения.

Преимущества и недостатки использования паттерна Observer

Преимущества использования паттерна Observer:

1. **Гибкость и расширяемость:** Паттерн Observer обеспечивает гибкую архитектуру, позволяя легко добавлять новых наблюдателей или издателей без изменения существующего кода.

2. **Разделение ответственностей:** Использование Observer позволяет разделить обязанности между издателями и наблюдателями, что способствует уменьшению связанности компонентов системы.

3. **Уменьшение зависимостей:** Наблюдатели не зависят от конкретной реализации издателя, что способствует созданию модульных компонентов, уменьшая прямые зависимости.

4. **Обновление по требованию:** Наблюдатели получают уведомления только при наличии соответствующих событий, что позволяет осуществлять обновление данных по требованию.

5. **Поддержка множественной рассылки:** Издатель способен уведомлять несколько наблюдателей одновременно, обеспечивая эффективную рассылку информации о событиях.

Недостатки использования паттерна Observer:

1. **Неявная зависимость:** В случае неаккуратной реализации, использование Observer может привести к неявным зависимостям между издателем и наблюдателем, усложняя понимание взаимосвязи между компонентами.

2. **Возможность утечек памяти:** при некорректном удалении наблюдателей из списка подписчиков у издателя может возникнуть утечка памяти, что требует аккуратного управления жизненным циклом объектов.

3. **Дополнительные вызовы методов:** из-за механизма уведомлений, наблюдатели могут получать лишние вызовы методов, если необходимая оптимизация не проведена правильно.

4. **Сложность отладки:** поскольку взаимодействие между издателем и наблюдателями происходит через уведомления, может быть более сложно отследить последовательность событий и действий в системе.

Хотя паттерн Observer предоставляет множество преимуществ, его использование требует внимательного подхода к проектированию и реализации, чтобы избежать вышеупомянутых недостатков.

5. Обзор существующих библиотек и фреймворков, использующих паттерн Observer в Python

1. **PyPubSub (Pypubsub)**: PyPubSub - это библиотека обмена сообщениями, которая обеспечивает реализацию паттерна Observer в Python. Она предоставляет механизм для передачи сообщений между различными компонентами приложения, где одни компоненты являются издателями, а другие – подписчиками. PyPubSub предлагает удобные API для создания каналов сообщений и подписки на них, что делает возможной эффективную обработку событий и уведомлений в приложениях.

2. **Django Signals**: Django Signals – это встроенный механизм веб-фреймворка Django, который позволяет отправлять сигналы из одной части приложения и принимать их в другой части. Сигналы в Django могут быть использованы для реализации паттерна Observer, где отправители сигналов выступают в роли издателей, а получатели – в роли подписчиков. Этот механизм обеспечивает эффективное взаимодействие между различными компонентами Django-приложений, позволяя им реагировать на изменения и события.

3. **RxPY (Reactive Extensions for Python)**: RxPY представляет собой библиотеку реактивного программирования, которая реализует реактивные расширения для языка Python. Она предоставляет мощный и удобный способ создания и управления потоками данных и событий, используя концепции наблюдателя и наблюдаемого, характерные для паттерна Observer. RxPY позволяет создавать сложные потоки данных, организовывать их обработку и реагировать на изменения в реальном времени, что делает её полезной для различных видов приложений, требующих обработки потоков данных.

Каждая из этих библиотек и фреймворков предоставляет удобные инструменты для использования паттерна Observer в Python, обеспечивая эффективное взаимодействие между компонентами приложений и обработку событий и уведомлений.

6. Сравнение паттерна Observer с другими подходами к обработке событий и уведомлений

Паттерн Observer и *подход на основе обратного вызова (Callback-based approach)*:

- **Observer**: Паттерн Observer устанавливает отношение зависимости "один ко многим" между издателями и подписчиками, позволяя автоматически реагировать на изменения. Издатель не знает о конкретных подписчиках, что делает систему более расширяемой и гибкой.
- **Callback-based approach**: В этом подходе, компоненты регистрируют функции обратного вызова (callbacks), которые будут вызываться при возникновении определенных событий. Основное различие заключается в том, что в случае обратного вызова, издатель должен явно вызывать соответствующие функции обратного вызова.

Паттерн Observer и *Централизованный шаблон обработки сообщений (Centralized Message Handling Pattern)*:

- **Observer**: предполагает, что издатель и подписчики независимы друг от друга, и подписчики получают уведомления об изменениях через интерфейс, предоставляемый издателем.
- **Централизованный шаблон обработки сообщений**: Здесь существует централизованный посредник, который получает все сообщения и распределяет их между заинтересованными компонентами. Это создает единую точку входа для управления сообщениями и событиями.

Паттерн Observer и *Публикация-подписка (Publish-Subscribe)*:

- **Observer**: реализует прямое уведомление подписчиков об изменениях в издателе.

- **Публикация-подписка**: В этом случае, издатель и подписчики не взаимодействуют напрямую, их связь устанавливается через посредника, называемого брокером сообщений, который отслеживает подписки и рассылает уведомления.

Паттерн Observer и *Реактивное программирование (Reactive Programming)*:

- **Observer**: предоставляет механизм, позволяющий издателям и подписчикам взаимодействовать независимо друг от друга. Издатель не имеет информации о конкретных подписчиках, что делает систему более расширяемой и гибкой.

- **Реактивное программирование**: включает в себя создание потоков данных и реакцию на изменения в этих потоках. Основными концепциями являются наблюдаемый объект (Observable) и наблюдатель (Observer), что соответствует паттерну Observer. Реактивное программирование предлагает более широкий спектр возможностей для обработки потоковых данных и асинхронных событий.

Паттерн Observer и *Использование глобальных переменных (Global Variables)*:

- **Observer**: фокусируется на установлении слабой связности между компонентами системы, позволяя издателям и подписчикам взаимодействовать независимо друг от друга.

- **Использование глобальных переменных**: может привести к жесткой зависимости между компонентами системы, так как глобальные переменные общие для всех компонентов. Это может усложнить разработку, тестирование и поддержку системы.

Паттерн Observer и *Шаблон проектирования Издатель-подписчик (Publisher-Subscriber)*:

- **Observer**: фокусируется на установлении отношения зависимости "один ко многим" между издателями и подписчиками, позволяя автоматически реагировать на изменения. Издатель не знает о конкретных подписчиках, что делает систему более расширяемой и гибкой.

- **Издатель-подписчик**: Этот шаблон также предназначен для обработки событий и уведомлений, и предполагает существование посредника (брокера), который управляет передачей сообщений между издателями и подписчиками. Подобно паттерну Observer, он обеспечивает слабую связность между компонентами.

Паттерн Observer и *Шаблон Обратное вызывание (Callback pattern)*:

- **Observer**: устанавливает отношение зависимости "один ко многим" между издателями и подписчиками, позволяя автоматически реагировать на изменения. Подписчики получают уведомления об изменениях в издателе через интерфейс, предоставляемый издателем.

- **Обратное вызывание**: В этом случае, компонент передает функцию другому компоненту, чтобы тот мог выполнить её в ответ на определенное событие.

7. Заключение.

Подведение итогов

Паттерн Observer, известный также как паттерн "Издатель-подписчик" (Publisher-Subscriber), представляет собой эффективный поведенческий паттерн проектирования, который устанавливает отношение зависимости "один ко многим" между объектами. Цель этого паттерна заключается в создании механизма подписки и оповещения, что позволяет объектам-наблюдателям автоматически реагировать на изменения, происходящие в объектах-субъектах.

Применение паттерна Observer способствует созданию слабосвязанных систем, где издатель и подписчики могут взаимодействовать без явных зависимостей друг от друга. Это обеспечивает гибкость, расширяемость и переиспользуемость кода, поскольку новые типы наблюдателей могут быть добавлены без изменения издателя, а изменения в издателе могут автоматически приводить к обновлению всех его наблюдателей.

В различных областях программирования, паттерн Observer находит широкое применение, включая пользовательские интерфейсы, системы управления базами данных, финансовые приложения и другие. Он обеспечивает эффективное управление изменениями и обновлениями в приложениях, позволяя обрабатывать события и уведомления.

Таким образом, паттерн Observer представляет собой мощный инструмент для создания гибких, расширяемых и эффективных систем, способствующих улучшению архитектуры приложений и обеспечивающих более простую обработку изменений в объектах.

Список литературы

1. Суманов В. "Python 3 и PyQt5. Разработка приложений". - БХВ-Петербург, 2019.
2. Федоренко В. "Python для сложных задач. Наука, бизнес, много данных". - ДМК Пресс, 2019.
3. Лутц М. "Изучаем Python". - Питер, 2016.
4. Терентьев В. Г. "Программирование на Python 3: подробное руководство". - ДМК Пресс, 2019.
5. Солянкин С., Чумаков А. "Python. Экспресс-курс". - Питер, 2019.