Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

Реферат на тему:

«Шаблонный метод: разработка гибких алгоритмов с использованием шаблонов в Python»

Выполнил:
Стригалов Дмитрий Михайлович

3 курс, группа ИВТ-б-о-21-1,

09.03.01 «Информатика и вычислительная техника», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:

Воронкин Р.А., канд. тех. наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь 2024

Оглавление

Введение

Шаблон проектирования в Python — это метод, который используется разработчиком для работы с часто встречающимся дизайном программного обеспечения. Проще говоря, это предопределенный шаблон для решения повторяющейся проблемы в коде. Эти шаблоны в основном разрабатываются на основе анализа требований.

Шаблон проектирования является частью разработки программного обеспечения. Это общее повторяемое решение потенциальной проблемы при разработке программного обеспечения. Мы можем следить за деталями шаблонов и применять решение, которое подходит для нашего кода.

Мы часто можем путать шаблоны и алгоритмы, но они являются отдельными подходами к решению повторяющихся задач. Алгоритмы обычно определяют четкий набор решений, которые могут быть реализованы в некоторых задачах, где шаблоны являются высокоуровневым описанием решения.

Например, алгоритм подобен кулинарному рецепту: у нас есть четкий набор ингредиентов (или набор решений), чтобы что-то приготовить (проблемы или цели). С другой стороны, шаблон подобен чертежу: мы можем видеть результат и его особенности, но можем изменить порядок реализации.
Позднее связывание приводит к одному следствию. Из базового класса можно вызывать методы и свойства, определенные в наследниках. Причем самих наследников может даже не существовать. Позднее связывание на то и позднее, что проверка происходит только в тот момент, когда этот код используется.
Эту особенность используют в паттерне «шаблонный метод». Он применяется, когда у подклассов есть общая логика, которая частично опирается на поведение подклассов. Такая логика реализуется в методе базового класса, а часть, которая различается для каждого подкласса, делегируется наследникам.

Шаблоны в языке программирования Python играют важную роль в различных областях разработки программного обеспечения. Вот несколько аспектов, которые демонстрируют их значение:

Упрощение и автоматизация кода:

– Шаблоны позволяют разработчикам упрощать процесс создания кода, предоставляя готовые к использованию каркасы или структуры.

– Возможность автоматической генерации кода с использованием шаблонов уменьшает рутинные задачи и ускоряет процесс разработки.

Web-разработка:

– В веб-разработке шаблоны часто используются для генерации динамических веб-страниц. Например, веб-фреймворки Django и Flask используют шаблоны для отображения данных на веб-страницах.

– Шаблонизаторы позволяют встраивать динамические данные в HTML, что облегчает разработку и обслуживание веб-приложений.

Генерация текстовых отчетов и документации:

– Шаблоны применяются для автоматической генерации текстовых отчетов, документации и других текстовых документов. Это позволяет быстро создавать структурированный и форматированный вывод.

Повторное использование кода:

Шаблоны способствуют повторному использованию кода, так как они позволяют выделить общие структуры и использовать их в различных частях программы или даже в разных проектах.

Механизмы шаблонов могут быть использованы для создания библиотек компонентов, которые можно легко внедрять в различные проекты.

Конфигурирование и настройка:

– Шаблоны применяются для создания конфигурационных файлов, позволяя пользователям легко настраивать поведение программы без необходимости изменения самого кода.

– Использование шаблонов упрощает создание и обслуживание файлов конфигурации.

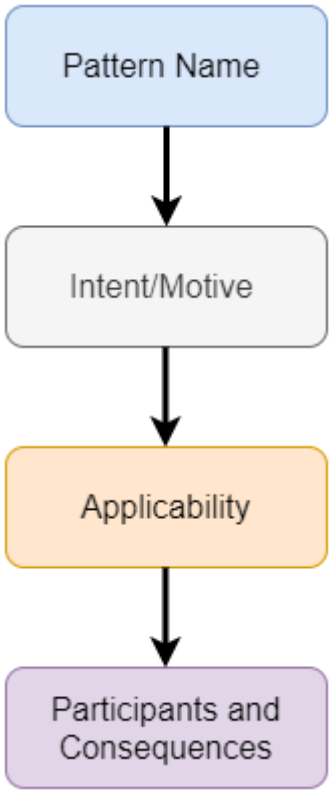
Работа с данными:

– Шаблоны используются для форматирования и представления данных в удобном для восприятия виде. Например, они могут применяться для форматирования строк, вывода данных в таблицы и т.д.

– Библиотеки для генерации отчетов или графиков часто используют шаблоны для определения внешнего вида и формата представления данных.

Конфигурация шаблона

На приведенной ниже диаграмме мы описываем базовую структуру документации шаблона проектирования. Он сосредоточен на том, какие технологии мы используем для решения проблем и какими способами.



На приведенной выше схеме:

Имя шаблона — используется для краткого и эффективного определения шаблона.

Намерение/мотив — определяет цель или то, что делает паттерн.

Применимость — определяет все возможные области, в которых шаблон применим.

Участники и последствия — состоит из классов и объектов, используемых в шаблоне проектирования, со списком последствий, существующих в шаблоне.

Преимущества использования шаблонов проектирования

Преимущества использования шаблонов проектирования приведены ниже:

Все шаблоны проектирования не зависят от языка.

Паттерны предлагают программистам выбрать проверенное решение для конкретных задач.

Они состоят из записи о выполнении, чтобы уменьшить любой технический риск для проектов.

Шаблоны просты в использовании и очень гибкие.

Важность изучения

Многие разработчики программного обеспечения могут работать в течение многих лет, не зная ни о какой закономерности. Также может случиться так, что мы реализуем шаблон, даже не подозревая об этом. Итак, здесь возникает вопрос, зачем нам изучать шаблоны проектирования? Давайте рассмотрим следующие моменты, которые освещают важность шаблонов проектирования в разработке.

Шаблоны проектирования имеют предопределенный набор испытанных и проверенных решений для общей проблемы, возникающей при разработке программного обеспечения. Если мы знаем о шаблоне проектирования, то можем применить решение, не теряя времени. Он также учит нас, как решить проблему, используя принцип объектно-ориентированного проектирования.

Шаблон проектирования также улучшает взаимопонимание между разработчиком и его товарищами по команде. Предположим, в коде есть проблема, и вы можете сказать «Используйте для этого Singleton», и каждый сможет понять, если знает шаблон проектирования и его название.

Шаблоны проектирования также полезны для целей обучения, потому что они представляют общую проблему, которую мы, возможно, игнорировали. Они также позволяют думать о той области, в которой ранее не было практического опыта.

Роль шаблонного метода в создании гибких алгоритмов

Шаблонный метод играет важную роль в создании гибких алгоритмов, предоставляя структуру, которая может быть легко адаптирована и расширена под конкретные требования. В этой главе рассмотрим, каким образом шаблонный метод способствует гибкости алгоритмов.

Абстракция и обобщение:

– Шаблонный метод позволяет создать абстракцию для общего алгоритма, выделяя ключевые шаги, которые могут быть реализованы в подклассах.

– Это способствует обобщению и выделению общей структуры, позволяя подклассам концентрироваться на своих уникальных особенностях.

Повторное использование кода:

– Шаблонный метод позволяет избежать дублирования кода, поскольку общая структура алгоритма выносится в родительский класс.

– Подклассы могут использовать готовый код родителя и сосредотачиваться только на том, что уникально для них.

Гибкость и изменение поведения:

– Шаблонный метод обеспечивает гибкость в изменении поведения алгоритма без изменения его структуры.

– Подклассы могут переопределять или расширять отдельные шаги алгоритма, не затрагивая другие его части.

Расширение функциональности:

– Позволяет добавлять новый функционал, внося изменения только в подклассы, сохраняя при этом целостность основной структуры.

– Это делает шаблонный метод мощным средством для расширения функциональности без изменения существующего кода.

Легкость поддержки и тестирования:

– Изменения в общем алгоритме, внесенные в родительский класс, автоматически применяются ко всем подклассам, что упрощает обслуживание кода.

– Тестирование отдельных шагов алгоритма проще, так как каждый шаг может быть протестирован независимо.

Преимущества использования шаблонных методов при разработке алгоритмов:

– Шаблонный метод определяет общую структуру алгоритма, обеспечивая согласованность в выполнении его шагов. Это особенно важно, когда нужно поддерживать консистентность в различных частях программы.

– Выделение общих шагов алгоритма в базовый класс позволяет повторно использовать код в различных частях программы. Это уменьшает дублирование кода и улучшает обслуживаемость.

– Шаблонные методы предоставляют гибкость в изменении отдельных шагов алгоритма в подклассах, сохраняя при этом общую структуру. Это позволяет легко адаптировать алгоритм под различные условия.

– Клиентский код, использующий шаблонный метод, взаимодействует с абстрактным базовым классом, что обеспечивает прозрачность и упрощает взаимодействие с алгоритмом. Клиент не заботится о деталях реализации шагов.

– В разработке крупных проектов шаблонные методы помогают структурировать код, делая его более понятным и легким для поддержки. Они позволяют разделить общие аспекты алгоритма от конкретных реализаций.

– Шаблонные методы обеспечивают простоту внесения изменений и расширения функциональности, поскольку каждый шаг алгоритма реализуется

в отдельном методе. Новые шаги можно легко добавлять без изменения общей структуры.

– Шаблонные методы хорошо интегрируются с принципами объектно-ориентированного программирования, такими как наследование и полиморфизм, что делает их естественным инструментом для проектирования алгоритмов в ООП-проектах.

– Применение шаблонных методов при разработке алгоритмов позволяет создавать структурированный, гибкий и легко поддерживаемый код, что делает их ценным инструментом в мире программирования.

Примеры использования шаблонного метода в создании гибких алгоритмов:

Примеры демонстрируют, как шаблонный метод может быть применен для создания гибких алгоритмов в различных областях программирования. Он позволяет легко расширять и изменять функциональность, обеспечивая при этом общую структуру алгоритма.

Веб-фреймворки (например, Django):

– В веб-разработке шаблонный метод используется для обработки запросов и формирования ответов.

– Различные этапы обработки запроса, такие как валидация, обработка данных, генерация ответа, могут быть реализованы в виде шагов шаблонного метода.

– Подклассы могут предоставлять конкретные реализации для обработки различных типов запросов.

Библиотеки обработки изображений (например, Pillow):

– Шаблонный метод может быть использован для создания алгоритма обработки изображений, включая операции, такие как изменение размера, фильтрация и сохранение.

– Различные подклассы могут предоставлять специфичные реализации для обработки разных форматов изображений.

Алгоритмы машинного обучения:

– В машинном обучении шаблонный метод может определить структуру обучения модели, включая предобработку данных, обучение модели и валидацию результатов.

– Подклассы могут настраивать процесс обучения для конкретных видов моделей или задач.

Алгоритмы сортировки:

– В алгоритмах сортировки шаблонный метод может определить общую структуру алгоритма (например, слияние или быстрая сортировка).

– Подклассы могут реализовывать специфичные шаги для оптимизации алгоритма под конкретные условия сортировки.

Фреймворки для тестирования (например, unittest в Python):

– Шаблонный метод может использоваться для создания общей структуры тестового кейса, включая установку, выполнение тестов и очистку.

– Подклассы могут определять конкретные тестовые случаи и ожидаемые результаты.

Примеры применения шаблонного метода

Пример использования шаблонного метода

Паттерн Шаблонный метод позволяет определить шаги алгоритма, оставив реализацию некоторых шагов подклассам.

Паттерн Шаблонный метод в некоторых отношениях похож на паттерн Мост.

```
"""
Шаблонный метод (Template method) - паттерн поведения классов.
Шаблонный метод определяет основу алгоритма и позволяет подклассам переопределить некоторые шаги алгоритма,
не изменяя его структуру в целом.
"""


class ExampleBase(object):
```

```
    def template_method(self):
        self.step_one()
        self.step_two()
        self.step_three()

    def step_one(self):
        raise NotImplementedError()

    def step_two(self):
        raise NotImplementedError()

    def step_three(self):
        raise NotImplementedError()


class Example(ExampleBase):
    def step_one(self):
        print 'Первый шаг алгоритма'

    def step_two(self):
        print 'Второй шаг алгоритма'

    def step_three(self):
        print 'Третий шаг алгоритма'


example = Example()
example.template_method()
```

Еще пример:

```
def get_text():
    return "plain-text"


def get_pdf():
    return "pdf"


def get_csv():
    return "csv"


def convert_to_text(data):
    print("[CONVERT]")
```

```
    return "{} as text".format(data)


def saver():
    print("[SAVE]")


def template_function(getter, converter=False, to_save=False):
    data = getter()
    print("Got `{}`".format(data))

    if len(data) <= 3 and converter:
        data = converter(data)
    else:
        print("Skip conversion")

    if to_save:
        saver()

    print("`{}` was processed".format(data))


def main():
    """
    >>> template_function(get_text, to_save=True)
    Got `plain-text`
    Skip conversion
    [SAVE]
    `plain-text` was processed
    >>> template_function(get_pdf, converter=convert_to_text)
    Got `pdf`
    [CONVERT]
    `pdf as text` was processed
    >>> template_function(get_csv, to_save=True)
    Got `csv`
    Skip conversion
    [SAVE]
    `csv` was processed
    """


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Оптимизация шаблонного метода в контексте класса

Возьмем для примера класс HTMLAnchorElement. Посмотрим на метод __str__(). Видно, что его код останется идентичным для большинства тегов. Единственное, что меняется, — название самого тега:

```
class HTMLAnchorElement(HTMLElement):
    def __str__(self):
        # Родительский метод
        attr_line = self.stringify_attributes()
        # Родительский метод
        body = self.get_text_content()
        return f'<a{attr_line}>{body}</a>'
```

Здесь мы видим пример класса HTMLAnchorElement, который наследуется от HTMLElement. Метод __str__() возвращает строку, представляющую HTML-элемент <a>, включающий атрибуты и текстовое содержимое.

Мы можем модифицировать код так, что метод __str__() переместится в HTMLElement. И единственная вещь, которая останется за подклассами, – имя тега:

```
class HtmlElement:

    def __str__(self):
        attr_line = self.stringify_attributes()
        body = self.get_text_content()
        tag_name = self.get_tag_name()
        return f"<{tag_name} {attr_line}>{body}</{tag_name}>"
```

В этом примере мы переместили общую логику по созданию HTML-элемента в базовый класс HtmlElement, а специфичные детали, например, имя тега, оставили за подклассами, которые должны реализовать метод get_tag_name().

Получившийся код лучше исходного варианта, так как он значительно сокращает дублирование (тегов около 100 штук).

В методе __str__ мы вызываем метод get_tag_name(), который должен быть реализован в наследниках:

```
class HtmlAnchorElement(HtmlElement):
    def get_tag_name(self):
        return "a"
```

Метод get_tag_name() в этом подклассе возвращает строку "a", которая является именем тега.

При этом теги бывают одиночные, значит, текущий вариант __str__ не подойдет для них. Из этой ситуации можно выйти разными способами, например, с помощью наследования

Концептуальный пример

Этот пример показывает структуру паттерна Шаблонный метод, а именно — из каких классов он состоит, какие роли эти классы выполняют и как они взаимодействуют друг с другом.

main.py: *Пример структуры паттерна*

```
from abc import ABC, abstractmethod



class AbstractClass(ABC):
    """
    Абстрактный Класс определяет шаблонный метод, содержащий скелет некоторого
    алгоритма, состоящего из вызовов (обычно) абстрактных примитивных операций.

    Конкретные подклассы должны реализовать эти операции, но оставить сам
    шаблонный метод без изменений.
```

```
    """

    def template_method(self) -> None:
        """
        Шаблонный метод определяет скелет алгоритма.
        """

        self.base_operation1()
        self.required_operations1()
        self.base_operation2()
        self.hook1()
        self.required_operations2()
        self.base_operation3()
        self.hook2()

    # Эти операции уже имеют реализации.

    def base_operation1(self) -> None:
        print("AbstractClass says: I am doing the bulk of the work")

    def base_operation2(self) -> None:
        print("AbstractClass says: But I let subclasses override some operations")

    def base_operation3(self) -> None:
        print("AbstractClass says: But I am doing the bulk of the work anyway")

    # А эти операции должны быть реализованы в подклассах.

    @abstractmethod
    def required_operations1(self) -> None:
```

```
        pass

    @abstractmethod
    def required_operations2(self) -> None:
        pass

    # Это «хуки». Подклассы могут переопределять их, но это не обязательно,
    # поскольку у хуков уже есть стандартная (но пустая) реализация. Хуки
    # предоставляют дополнительные точки расширения в некоторых критических
    # местах алгоритма.

    def hook1(self) -> None:
        pass

    def hook2(self) -> None:
        pass


class ConcreteClass1(AbstractClass):
    """
    Конкретные классы должны реализовать все абстрактные операции базового
    класса. Они также могут переопределить некоторые операции с реализацией по
    умолчанию.
    """

    def required_operations1(self) -> None:
        print("ConcreteClass1 says: Implemented Operation1")
```

```
    def required_operations2(self) -> None:
        print("ConcreteClass1 says: Implemented Operation2")


class ConcreteClass2(AbstractClass):
    """
    Обычно конкретные классы переопределяют только часть операций базового
    класса.
    """

    def required_operations1(self) -> None:
        print("ConcreteClass2 says: Implemented Operation1")

    def required_operations2(self) -> None:
        print("ConcreteClass2 says: Implemented Operation2")

    def hook1(self) -> None:
        print("ConcreteClass2 says: Overridden Hook1")


def client_code(abstract_class: AbstractClass) -> None:
    """
    Клиентский код вызывает шаблонный метод для выполнения алгоритма. Клиентский
    код не должен знать конкретный класс объекта, с которым работает, при
    условии, что он работает с объектами через интерфейс их базового класса.
    """

    # ...
```

```
    abstract_class.template_method()
    # ...



if __name__ == "__main__":
    print("Same client code can work with different subclasses:")
    client_code(ConcreteClass1())
    print("")

    print("Same client code can work with different subclasses:")
    client_code(ConcreteClass2())
```

Пример использования шаблонного метода в реальном проекте для обработки данных

Предположим, у нас есть проект по анализу данных, и нам нужно реализовать обработку данных разных типов (например, CSV и JSON). Мы можем использовать шаблонный метод для определения общей структуры обработки данных с возможностью расширения поддержки новых типов данных.

```
from abc import ABC, abstractmethod

# Базовый класс с шаблонным методом
class DataProcessor(ABC):
    def process_data(self, data):
        self.load_data(data)
        self.clean_data()
        self.transform_data()
        self.display_data()

    @abstractmethod
```

```
    def load_data(self, data):
        pass

    @abstractmethod
    def clean_data(self):
        pass

    @abstractmethod
    def transform_data(self):
        pass

    @abstractmethod
    def display_data(self):
        pass

# Подкласс для обработки CSV данных
class CSVDataProcessor(DataProcessor):
    def load_data(self, data):
        print(f"Loading CSV data: {data}")

    def clean_data(self):
        print("Cleaning CSV data")

    def transform_data(self):
        print("Transforming CSV data")

    def display_data(self):
        print("Displaying CSV data")

# Подкласс для обработки JSON данных
```

```
class JSONDataProcessor(DataProcessor):
    def load_data(self, data):
        print(f"Loading JSON data: {data}")

    def clean_data(self):
        print("Cleaning JSON data")

    def transform_data(self):
        print("Transforming JSON data")

    def display_data(self):
        print("Displaying JSON data")

# Пример использования
if __name__ == "__main__":
    csv_processor = CSVDataProcessor()
    json_processor = JSONDataProcessor()

    csv_processor.process_data("example.csv")
    print("------------------------")
    json_processor.process_data("example.json")
```

DataProcessor определяет шаблонный метод с четырьмя шагами обработки данных.

CSVDataProcessor и JSONDataProcessor предоставляют конкретные реализации шагов для обработки данных в форматах CSV и JSON соответственно.

При использовании подклассов происходит вызов шаблонного метода, который обеспечивает выполнение общей структуры алгоритма, при этом конкретные шаги реализуются в подклассах.

Пример разработки гибких алгоритмов с использованием шаблонов в Python. Допустим, у нас есть задача классификации данных, и нам нужно реализовать гибкий алгоритм обучения с использованием различных методов классификации. Мы можем использовать шаблонный метод для определения общей структуры алгоритма обучения, а подклассы будут реализовывать конкретные методы классификации.

```
from abc import ABC, abstractmethod

# Базовый класс с шаблонным методом
class ClassificationAlgorithm(ABC):
    def train(self, data):
        self.load_data(data)
        self.preprocess_data()
        self.extract_features()
        self.train_model()

    @abstractmethod
    def load_data(self, data):
        pass

    @abstractmethod
    def preprocess_data(self):
        pass

    @abstractmethod
    def extract_features(self):
        pass

    @abstractmethod
    def train_model(self):
```

```
        pass

# Подкласс для метода k-ближайших соседей
class KNNAlgorithm(ClassificationAlgorithm):
    def load_data(self, data):
        print(f"Loading data for KNN: {data}")

    def preprocess_data(self):
        print("Preprocessing data for KNN")

    def extract_features(self):
        print("Extracting features for KNN")

    def train_model(self):
        print("Training KNN model")

# Подкласс для метода опорных векторов
class SVMAlgorithm(ClassificationAlgorithm):
    def load_data(self, data):
        print(f"Loading data for SVM: {data}")

    def preprocess_data(self):
        print("Preprocessing data for SVM")

    def extract_features(self):
        print("Extracting features for SVM")

    def train_model(self):
        print("Training SVM model")
```

```
# Пример использования
if __name__ == "__main__":
    knn_classifier = KNNAlgorithm()
    svm_classifier = SVMAlgorithm()

    knn_classifier.train("classification_data.csv")
    print("------------------------")
    svm_classifier.train("classification_data.csv")
```

В этом примере:

ClassificationAlgorithm определяет шаблонный метод с четырьмя шагами обучения классификационной модели.

KNNAlgorithm и SVMAlgorithm предоставляют конкретные реализации шагов для методов k-ближайших соседей и опорных векторов соответственно.

При использовании подклассов происходит вызов шаблонного метода, обеспечивая выполнение общей структуры алгоритма обучения, при этом конкретные шаги реализуются в подклассах.

Применение шаблонных методов в различных сценариях

– Структура алгоритма остается постоянной, но некоторые шаги могут варьироваться: шаблонные методы полезны, когда у вас есть алгоритм, который должен сохранять определенную структуру, но некоторые его шаги могут различаться в зависимости от конкретной реализации.

– Повторное использование кода: когда вы хотите избежать дублирования кода, шаблонные методы предоставляют возможность выделить общую структуру в базовом классе, что упрощает повторное использование кода.

– Облегчение поддержки и расширения: шаблонные методы облегчают поддержку кода, поскольку изменения в общей структуре алгоритма автоматически применяются ко всем подклассам. Это также обеспечивает гибкость для расширения функциональности.

– Создание фреймворков и библиотек: при создании фреймворков и библиотек шаблонные методы могут быть использованы для определения общего интерфейса или структуры, который будет использоваться различными клиентами.

– Определение общего интерфейса в иерархии классов: когда необходимо определить общий интерфейс в иерархии классов, но оставить реализацию некоторых методов подклассам.

Ограничения применения шаблонных методов

– Алгоритм полностью изменяется в каждой подклассе: если каждый подкласс полностью переопределяет все методы шаблонного метода, то использование шаблонных методов может быть излишним.

– Когда структура алгоритма не является постоянной: если структура алгоритма часто меняется или зависит от условий выполнения, использование шаблонных методов может быть неэффективным.

– Сложность поддержки в случае большого числа подклассов: при большом количестве подклассов, особенно если у каждого из них есть свои уникальные особенности, управление иерархией классов может стать сложным.

– Ограничение гибкости и изменения: если требуется максимальная гибкость и возможность легкого изменения алгоритма, другие подходы, такие как стратегия или инъекция зависимостей, могут быть более подходящими.

– Когда у вас есть функциональные подходы: в функциональном программировании, где подход к построению алгоритмов может отличаться, шаблонные методы могут не быть самым естественным выбором.

Заключение

Шаблонный метод представляет собой мощный инструмент в разработке гибких алгоритмов на Python. На протяжении этого исследования мы рассмотрели его применение и сравнили с другими методами проектирования алгоритмов.

Шаблонный метод позволяет определить общую структуру алгоритма, сохраняя при этом гибкость в варьировании отдельных шагов. Это особенно полезно, когда необходимо обеспечить согласованность и повторное использование кода, а также способствует повторному использованию кода, выделяя общие шаги алгоритма в базовый класс. Это улучшает обслуживаемость кода и уменьшает дублирование.

В контексте объектно-ориентированного программирования шаблонный метод является удачным примером использования наследования для определения общего интерфейса и делегирования деталей реализации подклассам.

Примеры применения шаблонного метода в реальных проектах, таких как обработка данных и алгоритмы классификации, демонстрируют его эффективность в различных областях программирования.

В целом, шаблонный метод является ценным инструментом для создания гибких и легко поддерживаемых алгоритмов на Python, предоставляя элегантное решение для обеспечения согласованности и повторного использования кода в объектно-ориентированных проектах.

Список литературы

1. "Приемы объектно-ориентированного проектирования. Паттерны проектирования" Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес
2. "Head First Design Patterns" Эрик Фримен, Элизабет Фримен
3. Официальная документация Python.
4. Fluent Python" Люсиано Рамальо
5. "Design Patterns: Elements of Reusable Object-Oriented Software" Эрих
6. "Python Design Patterns" Артем Метельцев
7. "Effective Python: 90 Specific Ways to Write Better Python" Бретт Слаткин
8. "Refactoring: Improving the Design of Existing Code" Мартин Фаулер