

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ
КАФЕДРА ИНФОКОММУНИКАЦИЙ

РЕФЕРАТ

Тема:

«Паттерн Хранитель: реализация механизмов сохранения и восстановления состояния объектов в Python.»

Выполнил:

Лысенко Иван Александрович
студент 3 курса, группы ИВТ-б-о-21-1
направления подготовки /специальности
09.03.01 Информатика и вычислительная техника

(подпись)

Руководитель:

Воронкин Роман Александрович

(подпись)

Оценка _____

Ставрополь, 2023 г.

Оглавление

Введение	3
1. Общие сведения о паттерне «Хранитель» (Memento)	3
2. Структура паттерна «Хранитель»	4
3. Принципы реализации паттерна «Хранитель» в Python	6
4. Применение паттерна «Хранитель» в реальных проектах.....	8
5. Преимущества и недостатки при использовании паттерна «Хранитель»:.....	9
Заключение	11
Список литературы	12

Введение

Паттерны проектирования представляют собой проверенные и эффективные методики решения типовых проблем в процессе проектирования программного обеспечения. Они представляют собой обобщенные решения, разработанные и адаптированные для повторного использования в различных контекстах разработки. Использование паттернов способствует созданию гибкого, модульного и легко поддерживаемого кода.

Значение паттернов для разработки программного обеспечения нельзя переоценить. Они служат не только средством повышения эффективности и производительности, но и способом стандартизации подходов к решению конкретных задач. Паттерны обеспечивают понятность кода, снижают его сложность и уровень абстракции, что содействует легкости восприятия и сопровождения кодовой базы.

1. Общие сведения о паттерне «Хранитель» (Memento)

Паттерн «Хранитель» (или Memento) — это поведенческий паттерн, который обеспечивает механизм сохранения и восстановления состояния объектов без раскрытия их внутренней структуры. Он позволяет объектам сохранять свое текущее состояние и восстанавливать его в будущем, не раскрывая деталей своей реализации. Паттерн «Хранитель» часто используется для реализации механизмов отката (undo) и сохранения промежуточных состояний.

Паттерн «Хранитель» был впервые описан в книге «Design Patterns: Elements of Reusable Object-Oriented Software» (1994) Эрихом Гаммой, Ричардом Хелмом, Ральфом Джонсоном и Джоном Влиссидесом, также известными как «Банда четырех». Происхождение паттерна связано с потребностью разработки эффективных методов сохранения и восстановления состояния объектов в объектно-ориентированных приложениях.

Основные принципы и задачи, которые решает паттерн:

- Сохранение и восстановление состояния: Основной целью «Хранителя» является предоставление механизма для сохранения текущего состояния объекта и последующего его восстановления. Это позволяет объекту

восстанавливаться до предыдущих состояний без изменения своей внутренней структуры.

- Изоляция и прозрачность: Паттерн обеспечивает изоляцию состояния объекта от внешнего мира, предоставляя способ доступа к нему только через определенный интерфейс. Это обеспечивает прозрачность для внешних клиентов, которые могут работать с объектом, не зная деталей его реализации.

- Поддержка отката (undo) и восстановление: Паттерн «Хранитель» часто используется для реализации механизмов отката изменений и восстановления предыдущих состояний объектов. Это особенно полезно в приложениях, где важно иметь возможность отмены последних действий.

Паттерн «Хранитель» эффективно применяется в сценариях, где сохранение и восстановление состояния играют ключевую роль в обеспечении надежности и гибкости программного обеспечения.

2. Структура паттерна «Хранитель»

1. Originator (Источник):

Originator представляет объект, состояние которого должно быть сохранено и восстановлено. Его главной задачей является предоставление интерфейса для установки, получения и создания объектов Memento, содержащих его текущее состояние. Originator активно взаимодействует с Memento и является источником информации для сохранения.

Примеры сценариев использования Originator:

1. Редактор текста: В текстовом редакторе Originator может представлять собой объект текстового документа. При каждом изменении документа создается новый объект Memento, содержащий предыдущее состояние документа.

2. Графический редактор: В графическом редакторе Originator может быть объектом, представляющим рисунок или изображение. Создание моментов позволяет откатывать изменения и восстанавливать предыдущие версии рисунка.

2. Memento (Хранитель):

Memento — это объект-хранилище, который содержит снимок текущего состояния Originator. Его главной задачей является предоставление возможности Originator восстановить свое состояние. Memento должен быть спроектирован таким образом, чтобы он содержал достаточно информации для точного воссоздания состояния Originator.

Примеры информации, которую может содержать Memento:

1. Текстовый документ: Для редактора текста Memento может содержать текст документа, текущую позицию курсора и другие параметры форматирования.

2. Графический редактор: В графическом редакторе Memento может содержать данные о расположении объектов, их параметрах и других атрибутах.

3. Caretaker (Опекун):

Caretaker отвечает за управление объектами Memento и координирует их сохранение и восстановление. Его задачи включают в себя сохранение последних состояний Originator, управление списком Memento и предоставление Originator доступа к конкретным Memento для восстановления.

Примеры операций, выполняемых Caretaker:

1. Сохранение состояния: Caretaker сохраняет текущее состояние Originator, создавая новый объект Memento и добавляя его в свой список.

2. Восстановление состояния: Caretaker предоставляет Originator доступ к необходимому Memento для восстановления предыдущего состояния.

3. Управление списком Memento: Caretaker может реализовывать стратегии удаления устаревших состояний для оптимизации использования памяти.

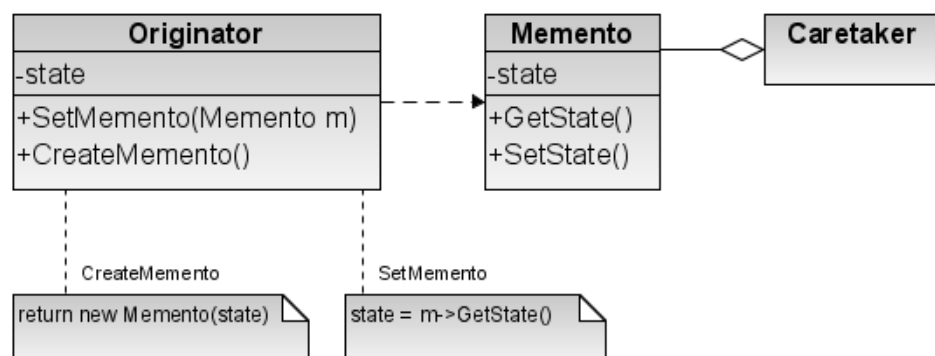


Рисунок 1 – Классический вариант шаблона хранитель

Классический вариант: Шаблон Хранитель используется двумя объектами: «Создателем» (originator) и «Опекуном» (caretaker). «Создатель» – это объект, у которого есть внутреннее состояние. Объект «Опекун» может производить некоторые действия с «Создателем», но при этом необходимо иметь возможность откатить изменения. Для этого «Опекун» запрашивает у «Создателя» объект «Хранителя». Затем выполняет запланированное действие (или последовательность действий). Для выполнения отката «Создателя» к состоянию, которое предшествовало изменениям, «Опекун» возвращает объект «Хранителя» его «Создателю». «Хранитель» является непрозрачным (то есть таким, который не может или не должен изменяться «Опекуном»).

Компоненты паттерна «Хранитель» тесно взаимодействуют для обеспечения сохранения и восстановления состояния объекта, при этом каждый компонент выполняет свою уникальную роль в этом процессе.

3. Принципы реализации паттерна «Хранитель» в Python

Пример базовой реализации класса Originator:

```
class Originator:
    def __init__(self):
        self.state = None

    def set_state(self, state):
        self.state = state

    def get_state(self):
        return self.state

    def create_memento(self):
        return Memento(self.state)

    def restore_from_memento(self, memento):
        self.state = memento.get_state()
```

Рисунок 2 – Класс Originator

В данной реализации класс «Originator» имеет атрибут «state», представляющий его текущее состояние. Метод «set_state» используется для установки нового состояния, а «get_state» — для получения текущего состояния. Метод

«create_memento» создает объект «Memento», содержащий текущее состояние «Originator». Метод «restore_from_memento» позволяет восстанавливать состояние объекта из переданного объекта «Memento».

Пример класса Memento и его взаимодействия с Originator:

```
class Memento:  
    def __init__(self, state):  
        self.state = state  
  
    def get_state(self):  
        return self.state
```

Рисунок 3 – Класс Memento

Класс «Memento» представляет объект, который содержит состояние «Originator». В данном примере он просто хранит состояние в атрибуте «state». Метод «get_state» используется для получения сохраненного состояния.

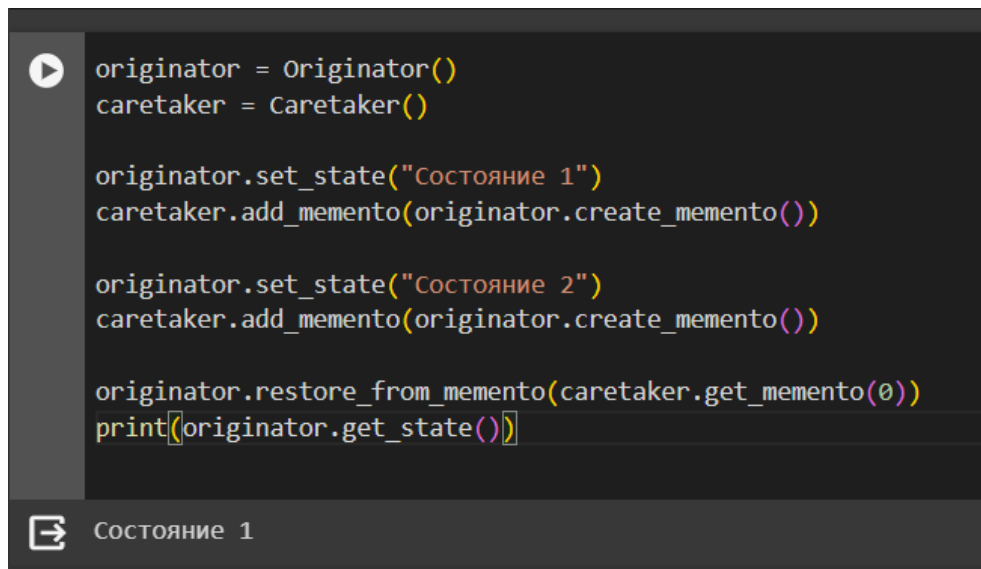
Пример класса Caretaker и его роли в управлении состоянием:

```
class Caretaker:  
    def __init__(self):  
        self.mementos = []  
  
    def add_memento(self, memento):  
        self.mementos.append(memento)  
  
    def get_memento(self, index):  
        return self.mementos[index]
```

Рисунок 4 – Класс Caretaker

Класс «Caretaker» является опекуном, ответственным за управление объектами «Memento». В данной реализации он хранит список объектов «Memento» в атрибуте «mementos». Метод «add_memento» добавляет новый объект «Memento» в список, а «get_memento» предоставляет доступ к конкретному объекту «Memento» по индексу.

Эти классы взаимодействуют между собой, обеспечивая сохранение, восстановление и управление состоянием объекта «Originator». Например:



```
originator = Originator()
caretaker = Caretaker()

originator.set_state("Состояние 1")
caretaker.add_memento(originator.create_memento())

originator.set_state("Состояние 2")
caretaker.add_memento(originator.create_memento())

originator.restore_from_memento(caretaker.get_memento(0))
print(originator.get_state())
```

Состояние 1

Рисунок 5 – Пример использования паттерна

Этот код демонстрирует создание объектов «Memento», сохранение и восстановление состояний с использованием классов «Originator» и «Caretaker».

4. Применение паттерна «Хранитель» в реальных проектах

1. Редактор текста или графический редактор:

Сценарий: В редакторе, при каждом изменении документа, может использоваться паттерн «Хранитель» для создания объекта «Memento», который содержит предыдущее состояние документа. Это позволяет пользователю отменять изменения и восстанавливать предыдущие версии.

2. Системы управления транзакциями:

Сценарий: В базах данных или системах управления транзакциями «Хранитель» может использоваться для сохранения состояния системы перед выполнением операций, таких как добавление, обновление или удаление записей. В случае ошибки или отката транзакции, состояние может быть восстановлено из объекта «Memento».

3. Игровые движки:

Сценарий: В игровых приложениях, особенно в ролевых играх, паттерн «Хранитель» может быть применен для сохранения состояния игры, позиции

персонажа, уровня и других параметров. Это обеспечивает возможность загрузки игры с определенного момента.

5. Преимущества и недостатки при использовании паттерна «Хранитель»:

Преимущества:

1. Отделение ответственности: Паттерн «Хранитель» позволяет разделить ответственность за сохранение и восстановление состояния от самого объекта. Это способствует более гибкому и чистому коду.

2. Поддержка отмены (undo) и восстановления: Паттерн обеспечивает прозрачный механизм для отмены последних действий и восстановления предыдущих состояний объекта.

3. Соккрытие деталей реализации: Использование «Хранителя» позволяет скрыть детали реализации объекта и его состояния от внешнего мира, предоставляя только необходимый интерфейс.

Недостатки:

1. Затраты на память: Хранение всех состояний может привести к затратам на память, особенно если система создает много объектов «Memento». Это может потребовать внимательного управления ресурсами.

2. Сложность в реализации: В больших проектах, особенно без должной архитектуры, внедрение «Хранителя» может быть нетривиальным и привести к усложнению кода.

Эффективность:

– Положительная эффективность: В сценариях, где важна поддержка отмены и восстановления, а также в управлении состоянием, применение паттерна «Хранитель» существенно улучшает структуру кода и обеспечивает надежность приложения.

– Ограниченная эффективность: В проектах с небольшим объемом состояний или где отмена действий не требуется, использование «Хранителя» может быть избыточным и привести к ненужной сложности.

Области применения:

- Редакторы и инструменты разработки: В приложениях для разработчиков, где важна история изменений и возможность отмены.
- Игровая индустрия: В разработке игр для сохранения и восстановления игровых состояний.
- Системы управления данными: В базах данных и системах управления транзакциями для обеспечения целостности данных.

В целом, паттерн «Хранитель» эффективно применяется в сценариях, где важно управление состоянием и обеспечение гибкости в отмене и восстановлении действий.

Заключение

В заключение, паттерн «Хранитель» предоставляет эффективный механизм для сохранения и восстановления состояния объектов в программных приложениях. Он отличается своей гибкостью и прозрачностью, позволяя отделить механизм сохранения состояния от основного объекта. В реферате были рассмотрены ключевые компоненты паттерна, его основные принципы и применение в реальных проектах.

Для разработчиков паттерн «Хранитель» представляет ценность в обеспечении поддержки отмены действий, сохранения состояний и улучшении общей надежности программного обеспечения. Отделение механизма сохранения от объекта позволяет создавать более гибкие и легко расширяемые системы. Кроме того, использование «Хранителя» снижает сложность кода, обеспечивая легкость поддержки и дальнейшего развития проекта.

Перспективы развития паттерна «Хранитель» остаются широкими. Дополнительные исследования могут быть направлены на:

1. Оптимизацию использования памяти: Разработка методов оптимизации хранения моментов для уменьшения нагрузки на память.
2. Интеграцию с современными технологиями: Исследование взаимодействия паттерна «Хранитель» с новыми технологиями, такими как асинхронное программирование или облачные вычисления.
3. Автоматизацию управления моментами: Разработка инструментов для автоматического управления моментами и оптимизации их хранения.

В целом, паттерн «Хранитель» остается актуальным и полезным инструментом в арсенале разработчиков, и его дальнейшее развитие может привести к ещё более эффективному использованию в различных областях программной инженерии.

Список литературы

1. Хранитель (шаблон проектирования), url:[https://ru.wikipedia.org/wiki/Хранитель_\(шаблон_проектирования\)](https://ru.wikipedia.org/wiki/Хранитель_(шаблон_проектирования))
2. Memento in Python, url:<https://refactoring.guru/design-patterns/memento/python/example>
3. Объясняю Pattern Memento (Снимок), url:<https://habr.com/ru/articles/689948/>
4. Implementation of Top Design Patterns in Python, url:<https://medium.com/python-pandemonium/top-design-patterns-in-python-9778843d5451>
5. Design Patterns in Python Object-Oriented Programming, url:<https://vegitbit.com/design-patterns-in-python-object-oriented-programming/>