СНИМОК

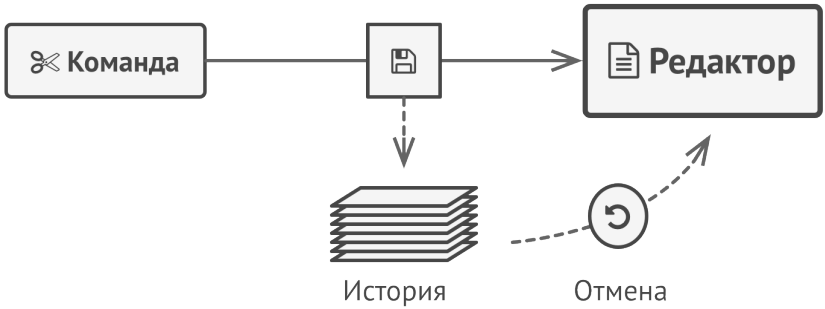
Также известен как: Хранитель, Memento

Снимок — это поведенческий паттерн проектирования, который позволяет делать снимки состояния объектов, не раскрывая подробностей их реализации. Затем снимки можно использовать, чтобы восстановить прошлое состояние объектов.

☹ Проблема

Предположим, что вы пишете текстовый редактор. Помимо обычного редактирования, ваш редактор позволяет менять форматирование текста, вставлять картинки и прочее.

В какой-то момент вы решили сделать все эти действия отменяемыми. Для этого вам нужно сохранять текущее состояние редактора перед тем, как выполнить любое действие. Если потом пользователь решит отменить своё действие, вы достанете копию состояния из истории и восстановите старое состояние редактора.



Перед выполнением команды, вы можете сохранить копию состояния редактора, чтобы потом иметь возможность отменить операцию.

Чтобы сделать копию состояния объекта, достаточно скопировать значение его полей. Таким образом, если вы сделали класс редактора достаточно открытым, то любой другой класс сможет заглянуть внутрь, чтобы скопировать его состояние.

Казалось бы, что ещё нужно? Ведь теперь любая операция сможет сделать резервную копию редактора перед своим действием. Но такой наивный подход обеспечит вам уйму проблем в будущем. Ведь если вы решите провести рефакторинг — убрать или добавить парочку полей в класс редактора — то придётся изменять код всех классов, которые могли копировать состояние редактора.

Как команде создать снимок состояния редактора, если все его поля приватные?

Но это ещё не все. Давайте теперь рассмотрим сами копии состояния. Из чего состоит состояние редактора? Даже самые примитивные редакторы требуют нескольких полей для хранения текущего открытого текста, позиции курсора и прокрутки экрана. Чтобы сделать копию состояния, вам нужно записать значения всех этих полей в некий «контейнер».

Скорее всего, вам понадобится хранить массу таких «контейнеров», поэтому удобней всего сделать их

объектами одного класса. Этот класс должен иметь массу полей и практически никаких методов. Чтобы другие классы смогли записывать и читать из него данные, вам придётся сделать его поля публичными. Но это приведёт к той же проблеме, что и с открытым классом редактора. Другие классы станут зависимыми от любых изменений в классе редактора.

Получается, нам придётся либо открывать классы для всех желающих, испытывая массу хлопот с поддержкой кода, либо делать классы закрытыми, но отказаться от идеи отмены операций. Нет ли какого-то другого пути?

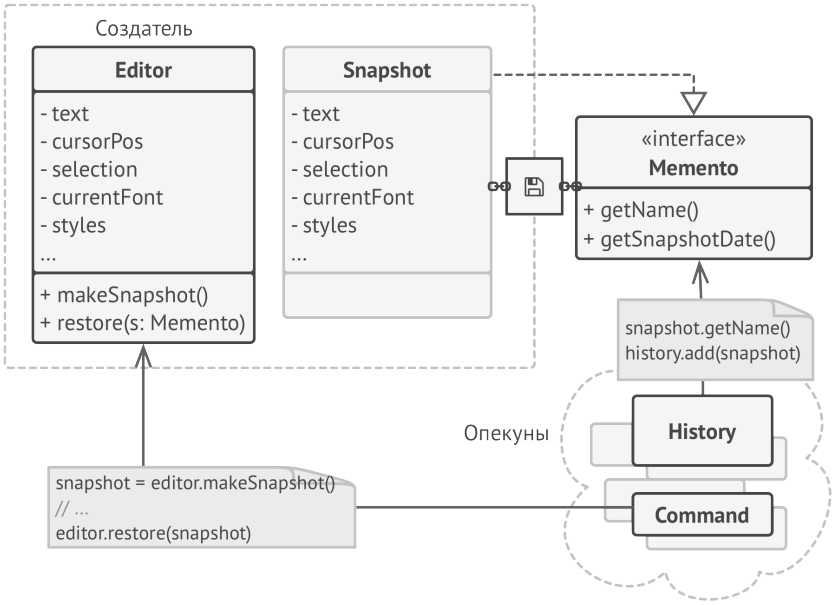
☺ Решение

Все проблемы, описанные выше, возникают из-за нарушения инкапсуляции. Это когда одни объекты пытаются сделать работу за других, влезая в их приватную зону, чтобы собрать необходимые для операции данные.

Паттерн Снимок поручает создание копии состояния объекта самому объекту, который этим состоянием владеет. Вместо того чтобы делать снимок «извне», наш редактор сам сделает копию своих полей — ведь ему доступны все поля, даже приватные.

Паттерн предлагает держать копию состояния в специальном объекте «*снимке*» с ограниченным

интерфейсом, позволяющим, например, узнать дату изготовления или название снимка. Но с другой стороны, снимок должен быть открыт для своего «*создателя*», позволяя прочесть и восстановить его внутреннее состояние.



Снимок полностью открыт для создателя, но лишь частично открыт для опекунов.

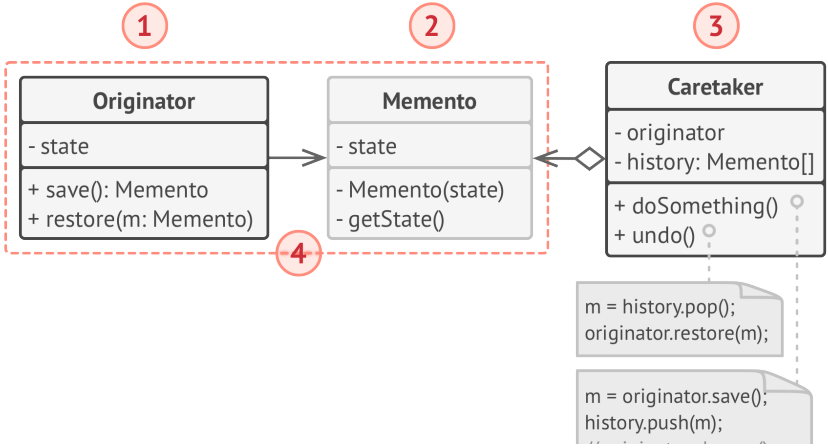
Эта схема позволяет создателям производить снимки и отдавать их для хранения другим объектам, называемым «*опекунами*». Опекунам будет доступен только ограниченный интерфейс снимка, поэтому они никак не смогут повлиять на внутренности самого снимка. В нужный момент, опекун может попросить создателя восстановить своё состояние, передав в него соответствующий снимок.

В нашем примере с редактором, опекуном можно сделать отдельный класс, который будет хранить список выполненных операций. Ограниченный интерфейс снимков позволит демонстрировать пользователю красивый список с названиями и датами выполненных операций. А когда пользователь решит откатить операцию, класс истории возьмёт последний снимок из стека и отправит его в редактор для восстановления.

Структура

Классическая реализация на вложенных классах

Классическая реализация паттерна полагается на механизм вложенных классов, которые доступны только в некоторых языках программирования (C++, C#, Java).



1. **Создатель** делает снимки своего состояния по запросу, а также воспроизводит прошлое состояние, если подать в него готовый снимок.

2. **Снимок** — это простой объект данных, содержащий состояние создателя. Надёжней всего сделать объекты снимков неизменяемыми и передавать в них состояние только через конструктор.

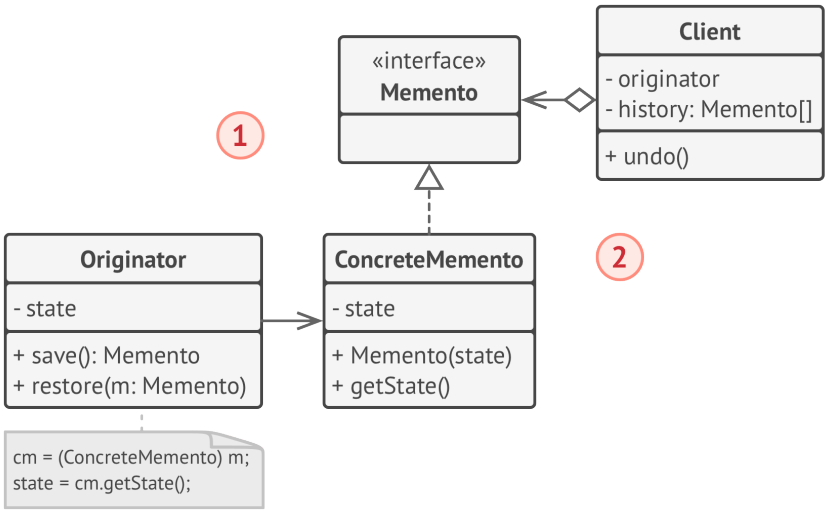
3. **Опекун** должен знать, когда делать снимок создателя и когда его нужно восстанавливать.

   Опекун может хранить историю прошлых состояний создателя в виде стека из снимков. Если понадобится сделать отмену, он возьмёт последний снимок и передаст его создателю для восстановления.

4. В этой реализации снимок — это внутренний класс по отношению к классу создателя, поэтому тот имеет полный доступ к его полям и методам, несмотря на то, что они объявлены приватными. Опекун же не имеет доступа ни к состоянию, ни к методам снимков и может всего лишь хранить ссылки на эти объекты.

Реализация с промежуточным пустым интерфейсом

Подходит для языков, не имеющих механизма вложенных классов (PHP).

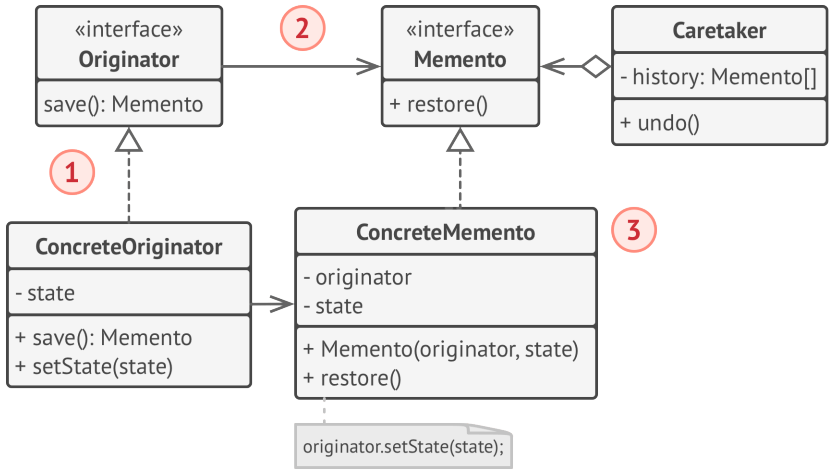


1. В этой реализации создатель работает напрямую с конкретным классом снимка, а опекун — только с его ограниченным интерфейсом.

2. Благодаря этому достигается тот же эффект, что и в классической реализации. Создатель имеет полный доступ к снимку, а опекун — нет.

Снимки с повышенной защитой

Когда нужно полностью исключить доступ к состоянию Создателей и Снимков.



1. Эта реализация разрешает иметь несколько видов создателей и снимков. Каждому классу создателей соответствует собственный класс снимков. Ни создатели, ни снимки не позволяют прочесть их состояние.

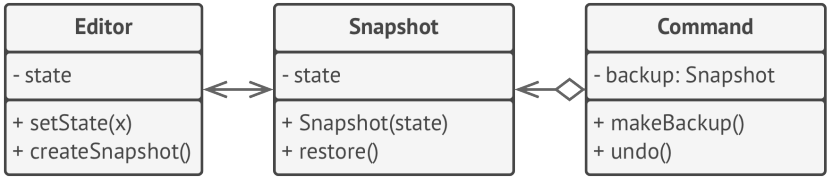
2. Здесь опекун ещё более жёстко ограничен в доступе к состоянию создателей и снимков. Но с другой стороны, опекун становится независим от создателей, так как метод восстановления теперь находится в самих снимках.

3. Снимки теперь связаны с теми создателями, из которых они сделаны. Они по-прежнему получают состояние через конструктор. Благодаря близкой связи между классами,

снимки знают, как восстановить состояние своих создателей.

Псевдокод

В этом примере паттерн **Снимок** используется совместно с паттерном **Команда** и позволяет хранить резервные копии сложного состояния текстового редактора и, если потребуется, восстанавливать его.



Пример сохранения снимков состояния текстового редактора.

Объекты команд выступают в роли опекунов и запрашивают снимки у редактора перед тем, как выполнить своё действие. Если потребуется отмена операции, команда сможет восстановить состояние редактора, используя сохранённый снимок.

При этом снимок не имеет публичных полей, поэтому никакой объект не сможет получить доступа его данным. Снимки связаны с определённым редактором, которых их создал и сам восстанавливает его состояние. Это позволяет программе иметь одновременно несколько объектов редакторов, например, разбитых по вкладкам.

```
// Класс создателя должен иметь специальный метод, который
// сохраняет состояние создателя в новом объекте-снимке.
class Editor is
  private field text, curX, curY, selectionWidth

  method setText(text) is
    this.text = text

  method setCursor(x, y) is
    this.curX = curX
    this.curY = curY

  method setSelectionWidth(width) is
    this.selectionWidth = width

  method createSnapshot(): EditorState is
    // Снимок — неизменяемый объект, поэтому Создатель
    // передаёт все своё состояние через
    // параметры конструктора.
    return new Snapshot(this, text, curX, curY, selectionWidth)

// Снимок хранит прошлое состояние редактора.
class Snapshot is
  private field editor: Editor
  private field text, curX, curY, selectionWidth

  constructor Snapshot(editor, text, curX, curY, selectionWidth) is
    this.editor = editor
    this.text = text
    this.curX = curX
    this.curY = curY
    this.selectionWidth = selectionWidth

  // В нужный момент, владелец снимка может восстановить
```

```
    // состояние редактора.
    method restore() is
      editor.setText(text)
      editor.setCursor(curX, curY)
      editor.setSelectionWidth(selectionWidth)

// Опекуном может выступать класс команд (см. паттерн
// Команда). В этом случае, команда сохраняет снимок
// получателя перед тем, как выполнить действие. А при
// отмене, возвращает получателя в предыдущее состояние.
class Command is
  private field backup: Snapshot

  method makeBackup() is
    backup = editor.saveState()

  method undo() is
    if (backup != null)
      backup.restore()
  // ...
```

Применимость

Когда вам нужно сохранять мгновенный снимок состояния объекта (или его части), чтобы впоследствии объект можно было восстановить в том же состоянии.

Паттерн Снимок позволяет делать любое количество снимков объекта и хранить их независимо от объекта, с которого делают снимок. Снимки часто используют не

только для реализации операции отмены, но и для транзакций, когда состояние объекта нужно откатить, если операция не удалась.

Когда прямое получение состояния объекта раскрывает детали его реализации и нарушает инкапсуляцию.

Паттерн предлагает изготовить снимок самому исходному объекту, так как ему доступны все поля, даже приватные.

Шаги реализации

1. Определите класс создателя, объекты которого должны создавать снимки своего состояния.

2. Создайте класс снимка и опишите в нём все те же поля, которые имеются в оригинальном классе-создателе.

3. Сделайте объекты снимков неизменяемыми. Они должны получать начальные значения только один раз, через свой конструктор.

4. Если ваш язык программирования это позволяет, сделайте класс снимка вложенным в класс создателя.

   Если нет, извлеките из класса снимка пустой интерфейс, который будет доступен остальным объектам программы. Впоследствии вы можете добавить некоторые

вспомогательные методы в этот интерфейс, дающие доступ к метаданным снимка, однако прямой доступ к данным создателя должен быть исключён.

5. Добавьте в класс создателя метод получения снимков. Создатель должен создавать новые объекты снимков, передавая значения своих полей через конструктор.

   Сигнатура метода должна возвращать снимки через ограниченный интерфейс, если он у вас есть. Сам класс должен работать с конкретным классом снимка.

6. Добавьте в класс создателя метод восстановления из снимка. Что касается привязки к типам, руководствуйтесь той же логикой, что и в пункте 4.

7. Опекуны, будь то история операций, объекты команд или нечто иное, должны знать о том, когда запрашивать снимки у создателя, где их хранить, и когда восстанавливать.

8. Связь опекунов с создателями можно перенести внутрь снимков. В этом случае каждый снимок будет привязан к своему создателю и должен будет сам восстанавливать его состояние. Но это будет работать либо если классы снимков вложены в классы создателей, либо если создатели имеют сеттеры для установки значений своих полей.

⚖ Преимущества и недостатки

✔ Не нарушает инкапсуляции исходного объекта.

✔ Упрощает структуру исходного объекта. Ему не нужно хранить историю версий своего состояния.

✖ Требует много памяти, если клиенты слишком часто создают снимки.

✖ Может повлечь дополнительные издержки памяти, если объекты, хранящие историю, не освобождают ресурсы, занятые устаревшими снимками.

✖ В некоторых языках (например, PHP, Python, JavaScript) сложно гарантировать, чтобы только исходный объект имел доступ к состоянию снимка.

⇄ Отношения с другими паттернами

- **Команду** и **Снимок** можно использовать сообща для реализации отмены операций. В этом случае объекты команд будут отображать выполненные действие над объектом, снимки — хранить копию состояния этого объекта до того, как команда была выполнена.

- **Снимок** можно использовать вместе с **Итератором**, чтобы сохранить текущее состояние обхода структуры данных и вернуться к нему в будущем, если потребуется.

- **Снимок** иногда можно заменить **Прототипом**, если объект, чьё состояние требуется сохранять в истории, довольно простой, не имеет активных ссылок на внешние ресурсы, либо их можно легко восстановить.