Wzorce projektowe - Memento

Inżynieria Systemów dla Fizyków

Gabriela Godek, Gabriela Białoskórska, Wojciech Kura, Ignacy Tekieli
Fizyka Techniczna III.

Czerwiec 2021

Spis treści

1 Wprowadzenie **2**
1.1 Cel pracy 2
1.2 Wstęp teoretyczny 2
1.2.1 Wzorce projektowe 2
1.2.2 Memento 3

2 Materiały i Metody **4**
2.1 Wykorzystane narzędzia 4
2.2 Implementacja 4
2.2.1 Główny kod programu 4
2.2.2 Pozostałe pliki repozytorium 8

3 Wyniki **9**

4 Podsumowanie **10**

5 Bibliografia **10**

1 Wprowadzenie

1.1 Cel pracy

Celem pracy było wykorzystanie wybranego wzorca projektowego do wykonania określonego projektu. W procesie tworzenia należało użyć systemu kontroli wersji oraz wdrożyć testy. Niniejsza praca ukazuje zastosowanie wzorca *Memento*, bazując na edytorze tekstu.

1.2 Wstęp teoretyczny

1.2.1 Wzorce projektowe

Wzorce projektowe to typowe rozwiązania często występujących problemów w projektowaniu oprogramowania. Są jak gotowe plany, które można dostosować, aby rozwiązać powtarzający się błąd projektowy w kodzie.

Należy pamiętać, że wzorzec nie jest konkretnym fragmentem kodu, ale ogólną koncepcją rozwiązania danego problemu. Nie można go po prostu skopiować i wkleić do programu, tak jak w przypadku bibliotek lub gotowych funkcji. Można za to śledzić szczegóły wzorca i wdrażać rozwiązania dostosowane do realiów własnej aplikacji.

Wzorce projektowe a algorytmy
Wzorce są również często mylone z algorytmami; obie koncepcje stanowią odpowiedź na powtarzalne błędy. Algorytm zawsze definiuje jasny zestaw działań, które umożliwiają osiągnięcie danego celu, natomiast wzorce są bardziej ogólnym opisem rozwiązania. Kod tego samego wzorca zastosowany do dwóch różnych programów może się znacznie różnić, ponadto w przypadku wzorca znamy jego wynik i cechy, ale kolejność wdrażania zależna jest od nas.

Z czego składa się wzorzec projektowy?
Większość wzorców posiada formalny opis, dzięki czemu każdy może odtworzyć ich ideę w różnych kontekstach. Oto sekcje na które zwykle dzieli się opis wzorca:

- **Cel** pobieżnie opisuje zarówno problem, jak i rozwiązanie.
- **Motywacja** rozszerza opis problemu i rozwiązania jakie umożliwia dany wzorzec.
- **Struktura** klas ukazuje poszczególne części wzorca i jak są ze sobą powiązane.
- **Przykład kodu** w którymś z popularnych języków programowania pomaga zrozumieć ideę wzorca.

1.2.2 Memento

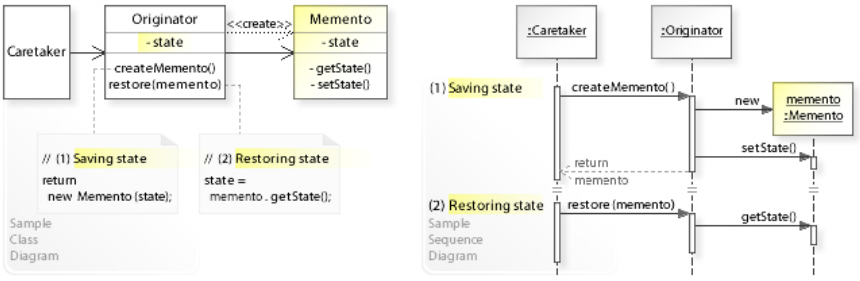
Memento to behawioralny wzorzec projektowy pozwalający zapisywać i przywracać wcześniejszy stan obiektu bez ujawniania szczegółów jego implementacji.

Niektóre obiekty próbują robić więcej niż powinny. Aby zbierać dane w celu wykonania jakiegoś zadania, wkradają się w prywatną przestrzeń innych obiektów, zamiast pozwolić im na samodzielne wykonanie tego zadania. Memento deleguje tworzenie migawki stanu samemu właścicielowi stanu — obiektowi *źródło*. Dlatego też, zamiast pozwalać innym obiektom próbować skopiować stan edytora "z zewnątrz", sama klasa edytora może wykonać migawkę siebie, gdyż ma pełny dostęp do swojego stanu.

Wzorzec proponuje przechowywanie kopii stanu w specjalnym obiekcie zwanym *pamiątką*. Zawartość pamiątki nie jest dostępna innym obiektom, oprócz jej twórcy. Muszą one komunikować się z pamiątką za pośrednictwem ograniczonego interfejsu, który pozwala na pobieranie metadanych migawki (czas utworzenia, nazwa wykonanej operacji, itd.), ale nie stanu pierwotnego obiektu zawartego w migawce.

Zastosowanie Memento
Memento może zostać wykorzystane w procesorze tekstu do zaimplementowania operacji "Cofnij" oraz "Ponów". Za każdym razem kiedy użytkownik wykonuje jakąś akcję – wprowadza tekst, zmienia wielkość czcionki czy jej kolor – tworzony jest obiekt pamiątki zapamiętujący bieżący stan dokumentu. Gdy użytkownik zleci wycofanie ostatniej operacji, stan dokumentu zostanie odtworzony za pomocą wcześniej zapisanej 'pamiątki'.

Konsekwencje stosowania
Jedną z konsekwencji stosowania tego wzorca jest umożliwienie zachowania hermetyzacji obiektu dla którego tworzona jest pamiątka. Jedną z wad Memento jest jego kosztowność w kwestii wykorzystywanej pamięci.



Rysunek 1: Przykładowy diagram klas i sekwencji UML dla wzorca projektowego Memento

2 Materiały i Metody

2.1 Wykorzystane narzędzia

W niniejszej pracy wykorzystano wzorzec *Memento* do symulacji edytora tekstu, konkretnie funkcji "Cofnij" oraz "Ponów". W tym celu posłużono się następującymi narzędziami:

- **GitHub** - hostingowy serwis internetowy, przeznaczony dla projektów programistycznych, wykorzystujących system kontroli wersji Git. W powyższym projekcie znacznie ułatwił on pracę grupową. Link do repozytorium projektu: `https://github.com/GabiBia/DesignPattern-Memento`
- **Doxygen** - generator dokumentacji, w przypadku powyższego projektu dla języka Python. Doxyfile również znajduje się w repozytorium.
- **Umbrello** - darmowy program komputerowy służący do tworzenia diagramów UML, dostępny dla systemów typu Unix. Jest częścią środowiska graficznego KDE i udostępniany jest na licencji GNU General Public License. Umożliwił stworzenie diagramu (rys. 2)
- **Make** - program powłoki systemowej automatyzujący proces kompilacji programów.
- **Jira** - zamknięte oprogramowanie firmy Atlassian, służące do śledzenia błędów oraz zarządzania projektami. Umożliwiło ono sprawną pracę nad projektem przy wykorzystaniu metodologii Scrum.

2.2 Implementacja

2.2.1 Główny kod programu

Główny kod programu napisany został w języku Python. Każdy jego fragment jest czytelnie opisany, tak aby można było zrozumieć zasadę jego działania.

Implementacja:

```
from __future__ import annotations
from abc import ABC, abstractmethod
from datetime import datetime
from random import sample
from string import ascii_letters, digits

class Inicjator():

    """
    Inicjator zawiera wazne stany, ktore moga ulec zmianie
    z biegiem czasu.
    Definiuje on rowniez metode zapisywania stanu w memento,
```

```
a takze metode odzyskiwania tegoz stanu z memento.
"""

_state = None

"""
W celu uproszczenia, stan Inicjatora jest przechowywany
w pojedynczej zmiennej.
"""

def __init__(self, state: str) -> None:
    self._state = state
    print(f"Inicjator: Moj stan poczatkowy to: {self._state}")

def do_something(self) -> None:

    """
    Dzialanie Inicjatora moze miec wplyw na jego wewnetrzny stan,
    dlatego tez powinien on zostac *zbackupowany* poprzez save()
    zanim uruchomione zostanie dzialanie zmieniajace stan.
    """

    print("Inicjator: Pracuje...")
    self._state = self._generate_random_string(30)
    print(f"Inicjator: moj stan zostal zmieniony: {self._state}")

def _generate_random_string(self, length: int = 10) -> None:
    return "".join(sample(ascii_letters, length))

def save(self) -> Memento:

    """
    Zapisuje obecny stan w memento
    :return:
    """

    return ConcreteMemento(self._state)

def restore(self, memento: Memento) -> None:
    """
    Przywraca stan inicjatora z memento
    :param memento:
    :return:
    """

    self._state = memento.get_state()
```

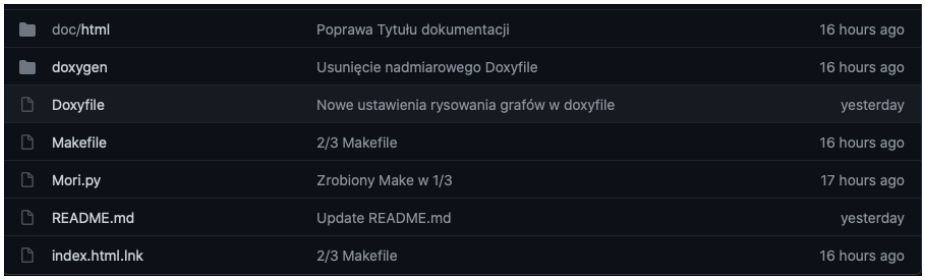
```
        print(f"Inicjator: moj stan zostal zmieniony: {self._state}")


class Memento(ABC):

    """
    Interfejs Memento zapewnia sposoby odzyskania danych
    szczegolnych memento, takich jak data utworzenia
    czy nazwa. Mimo to nie ujawania on stanu Inicjatora.
    """

    @abstractmethod
    def get_name(self) -> str:
        pass

    @abstractmethod
    def get_date(self) -> str:
        pass


class ConcreteMemento(Memento):
    def __init__(self, state: str) -> None:
        self._state = state
        self._date = str(datetime.now())[:19]

    def get_state(self) -> str:
        """
        Inicjator u ywa tej metody gdy przywraca jaki  stan.
        :return:
        """
        return self._state
    def get_name(self) -> str:
        """
        Reszta metod jest uzywana przez Opiekuna by wyswietlic szczegoly.
        :return:
        """
        return f"{self._date} / ({self._state[0:9]}...)"

    def get_date(self) -> str:
        return self._date


class Opiekun():

    """
    Opiekun nie polega na klasie Concrete Memento. Dlatego nie musi miec
```

```
    dostepu do stanu Inicjatora przechowywanego w memento.
    Dziala ze wszystkimi memento poprzez podstawowy interferjs Memento.
    """

    def __init__(self, inicjator: Inicjator) -> None:
        self._mementos = []
        self._inicjator = inicjator

    def backup(self) -> None:
        print("\nOpiekun: Zapisuje stan Inicjatora...")
        self._mementos.append(self._inicjator.save())

    def undo(self) -> None:
        if not len(self._mementos):
            return

        memento = self._mementos.pop()
        print(f"Opiekun: Przywracam stan do: {memento.get_name()}")
        try:
            self._inicjator.restore(memento)
        except Exception:
            self.undo()

    def show_history(self) -> None:
        print("Opiekun: Oto lista zapisanych Memento:")
        for memento in self._mementos:
            print(memento.get_name())

if __name__ == "__main__":

    inicjator = Inicjator("Lorem ipsum dolor sit posuere.")
    opiekun = Opiekun(inicjator)


opiekun.backup()
inicjator.do_something()

opiekun.backup()
inicjator.do_something()

opiekun.backup()
inicjator.do_something()

print()
opiekun.show_history()
```

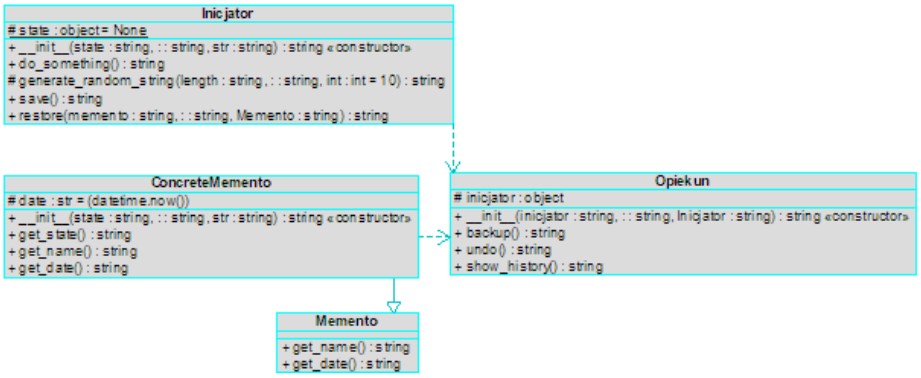
```
print("\nCTRL+Z\n")
opiekun.undo()

print("\nCTRL+Z\n")
opiekun.undo()

print("\nCTRL+Z\n")
opiekun.undo()

input("\nKoniec: nacisnij ENTER aby wyjsc.")
```

2.2.2 Pozostałe pliki repozytorium

Oprócz głównego kodu, w repozytrium projektu znajduje się również `Doxyfile` (oraz foldery z nim związane), a także `Makefile` i `README.md`. Cel ich zastosowania opisany został w rozdziale 2.1. Całość repozytorium dostępna jest w linku: `https://github.com/GabiBia/DesignPattern-Memento`



Dodatkowo, na potrzeby projektu, wykonano diagram klas *Memento*, korzystając z Umbrello:



Rysunek 2: Diagram klas wzorca Memento

3 Wyniki

Gdy użytkownik wywołuje cofnięcie operacji, historia pobiera najnowszą pamiątkę ze stosu i przekazuje ją edytorowi z żądaniem cofnięcia. Edytor ma pełny dostęp do pamiątki, zatem zmienia swój stan w oparciu o wartości w niej zawarte.

```
Inicjator: Moj stan poczatkowy to: Lorem ipsum dolor sit posuere.

Opiekun: Zapisuje stan Inicjatora...
Inicjator: Pracuje...
Inicjator: moj stan zostal zmieniony: gBxAvFbNkTqUQuJElYtSpGLcnjoXia

Opiekun: Zapisuje stan Inicjatora...
Inicjator: Pracuje...
Inicjator: moj stan zostal zmieniony: JXVxlmnWLYdKkCBwOAsztHuoeUIpyv

Opiekun: Zapisuje stan Inicjatora...
Inicjator: Pracuje...
Inicjator: moj stan zostal zmieniony: KwejiEVbOqfHFdacyMQsJADgZmpNGP

Opiekun: Oto lista zapisanych Memento:
2021-06-17 21:19:49 / (Lorem ips...)
2021-06-17 21:19:49 / (gBxAvFbNk...)
2021-06-17 21:19:49 / (JXVxlmnWL...)

CTRL+Z

Opiekun: Przywracam stan do: 2021-06-17 21:19:49 / (JXVxlmnWL...)
Inicjator: moj stan zostal zmieniony: JXVxlmnWLYdKkCBwOAsztHuoeUIpyv

CTRL+Z

Opiekun: Przywracam stan do: 2021-06-17 21:19:49 / (gBxAvFbNk...)
Inicjator: moj stan zostal zmieniony: gBxAvFbNkTqUQuJElYtSpGLcnjoXia

CTRL+Z

Opiekun: Przywracam stan do: 2021-06-17 21:19:49 / (Lorem ips...)
Inicjator: moj stan zostal zmieniony: Lorem ipsum dolor sit posuere.
```

Rysunek 3: Wynik urochomienia kodu

4 Podsumowanie

Wykonana symulacja funkcji edytora tekstu umożliwia najprostsze i najbardziej dokładne przedstawienie zasady działania wzorca *Memento*. Jego niewątpliwą zaletą jest możliwość tworzenia migawki stanu obiektów bez naruszania ich hermetyzacji. Ponadto dzięki zastosowaniu Memento możemy uprościć kod źródła, umożliwiając śledzenie historii stanu źródła. Mimo to większość dynamicznych języków programowania, jak PHP, Python i JavaScript, nie daje gwarancji niezmienialności stanu pamiątki, a aplikacja może wymagać zbyt dużej ilości pamięci RAM, jeśli klienci zbyt często będą je pamiątki.

5 Bibliografia

Materiały wykorzystane do napisania programu oraz wstępu:

- `https://en.wikipedia.org/wiki/Memento_pattern`
- `https://refactoring.guru/design-patterns/memento`

Repozytorium projektu:

- `https://github.com/GabiBia/DesignPattern-Memento`