

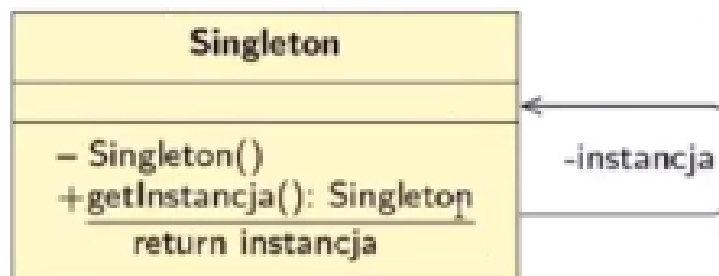
Contents

1 wzorce kreacyjne	2
1.1 singleton	2
1.1.1 implementacja	2
1.2 iterator	3
2 wzorce behawioralne	3
2.1 Obserwator	3
2.1.1 kontekst	4
2.1.2 problem	4
2.1.3 implementacja	4
2.2 Stan	4
2.2.1 kontekst	4
2.2.2 problem	5
2.2.3 implementacja	5
2.3 strategia	5
3 wzorce strukturalne	6
3.1 kompozyt	6
3.1.1 kontekst	6
3.1.2 problem	6



1 wzorce kreacyjne

1.1 singleton



Rysunek 9: Schemat wzorca *Singleton*

- zagwarantować że jest jeden obiekt tego typu (np. konfiguracja/stan globalny)

1.1.1 implementacja

```

class singleton {

private static singleton; //nasz obiekt
  
```

```

public static singleton getSingleton() //statyczna publiczna funkcja do otrzymywania t
{
    if(instancja==null)
        instancja = new Singleton();

    return singleton;
}
};

```

1.2 iterator

- hermetyzacja iteracji

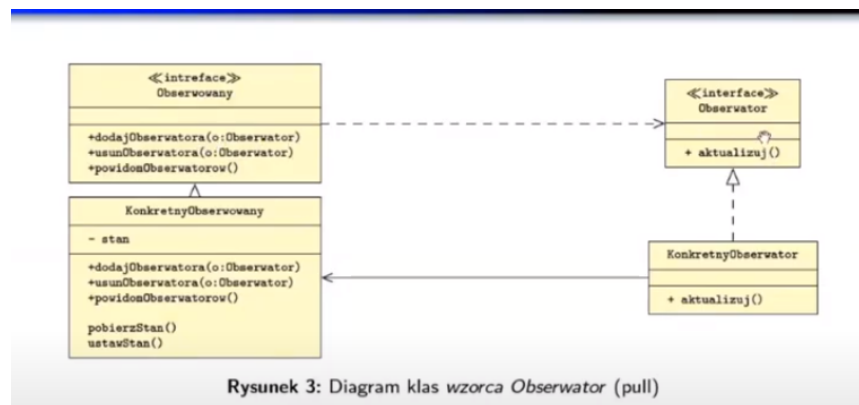
```

Iterator iterator = menuCostam.utworzIterator();
while (iterator.hasNext())
{
    pozycjaMenu pozycja = iterator.next();
}

```

2 wzorce behawioralne

2.1 Obserwator



- określa zależność jeden do wiele między obiektami
- gdy jeden obiekt zmienia stan wszystkie obiekty od niego zależne są o tym automatycznie powiadamiane i uaktualniane (np. w kalkulatorze)

mamy 3 klasy wypisywania ktore maja w sobie string do wypisywania, kiedy wprowadzamy nowe dzialanie wszystkie sa updatowane)

- wydaje mi sie ze realizowany w grach -> bo trzeba updatowac stan obiektow a one musza znac stan innych

2.1.1 kontekst

zmiana stanu jednego obiektu wymaga zmiany innych i nie wiadomo, ile obiektow trzeba zmienic

2.1.2 problem

obiekt powinien byc w stanie powiadamiac inne obiekty, nie przyjmujac zadnych zalozen co do tego, co te obiekty reprezentuja - wynikiem sa luzniejsze powiazania miedzy obiektami

2.1.3 implementacja

<https://refactoring.guru/design-patterns/observer> zagwarantowanie ze przed rozeslaniem powiadomienia stan obserwowanego jest wewnetrznie spojny

model push (obserwowany wysyla wszystkie informacje same) model pull (obserwowany wysyla POWIADOMIENIE a kazdy inny pyta sie to czego potrzebuje z jakiej zmiany)

2.2 Stan

- umozliwia obiektowi zmiane zachowania, gdy zmienia sie jego stan wewnetrzny (np. ktos zmienia typ konta bankowego)

2.2.1 kontekst

- zachowanie obiektu zalezy od jego stanu, a obiekt ten musi zmieniac swoje zachowanie w czasie wykonywania programu w zaleznosci od stanu
- operacje zawieraja duze, wieloczesciowe instrukcje warunkowe ktore zaleza od stanu obiektu - wzorzec State przenosi kazde rozgalezienie do specjalnej klasy z inna implementacja np. pobierz podatek

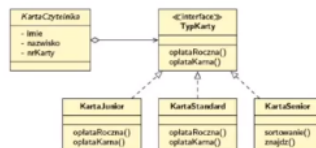
2.2.2 problem

chcemy umożliwić obiektowi zmianę zachowania w momencie zmiany wewnętrznego stanu obiektu hermetyzując stan w postaci klasy

2.2.3 implementacja

ROZWIĄZANIA

Drugie rozwiązanie polega na rozdzieleniu odpowiedzialności *Karty Czytelnika* na część przechowującą dane i część reprezentującą stan.



Rysunek 6: Drugi przykład rozwiązania problemu typu kart czytelnika

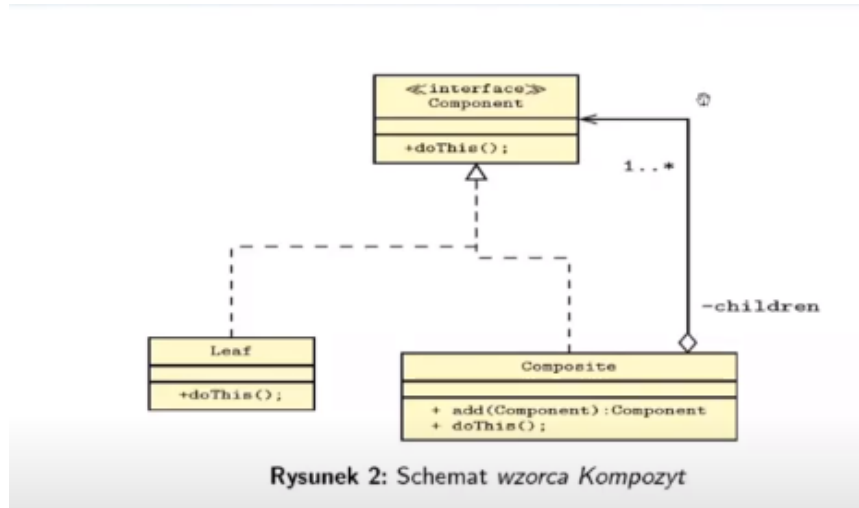
- Część przechowująca dane, nadal nazywana *Kartą Czytelnika*, posiada referencję do obiektu reprezentującego aktualny typ, dziedziczącego po klasie abstrakcyjnej lub implementującej interfejs. Dzięki temu zmiana typu wymaga jedynie utworzenia instancji innej klasy *Typ Karty* i przypisanie jej do *Karty Czytelnika*.
- Efektem takiego projektu jest czytelniejszy podział odpowiedzialności, który jednocześnie posiada zalety brakujące w poprzednim rozwiązaniu.

2.3 strategia

-

3 wzorce strukturalne

3.1 kompozyt



Rysunek 2: Schemat wzorca Kompozyt

TLDR: Drzewko w którym liść zawiera siebie + listę dzieci

- zadaniem jest łączenie obiektów w strukturę tak, że reprezentują hierarchie części-całości, unifikując dostęp do kolekcji jak i pojedynczego obiektu.
- + umożliwia to klientom jednolite traktowanie pojedynczych obiektów i również ich kompozycji

3.1.1 kontekst

chcemy przedstawić hierarchie obiektów część-całość Hierarchia obiektów ma wspólną klasę bazową (abstrakcyjną)

3.1.2 problem

chcemy, aby klienci mogli ignorować różnice między złożeniami obiektów a pojedynczymi obiektami - klienci będą wtedy jednakowo traktować wszystkie obiekty występujące w strukturze