Studium zastosowania wzorców czynnościowych w projekcie platformy sprzedaży typu Allegro

# 1. Wzorzec: Command

### 1.1 Motywacja

W projekcie platformy sprzedaży typu Allegro występuje wiele operacji, które reprezentują konkretne akcje biznesowe, takie jak wystawienie produktu na sprzedaż **submit()**, usunięcie ogłoszenia **delete()**, czy dokonanie zakupu **buy()** . Zastosowanie wzorca Command pozwala ujednolicić interfejs dla tych operacji, oddzielić logikę wykonania od wywołania oraz wprowadzić takie funkcjonalności jak historia operacji, kolejki zadań czy cofanie operacji.

**Konsekwencje zastosowania wzorca:**

* Wzrost elastyczności systemu.
* Umożliwość logowania operacji i ich wykonywania w późniejszym czasie.
* Oddzielenie obiektów wywołujących polecenia od tych, które je wykonują.

### 1.2 Struktura przed zastosowaniem wzorca

Przed wdrożeniem wzorca metoda **submit()** znajdowała się bezpośrednio w klasie **Item**:Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

### 1.3 Struktura po zastosowaniu wzorca

Po zastosowaniu wzorca dodano abstrakcyjną klasę **Command** oraz klasę **SubmitItemCommand**, która implementuje interfejs wykonania operacji:

Obraz zawierający tekst, linia, woda, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.

### 1.4 Współdziałanie po zastosowaniu wzorca

Podczas wystawiania przedmiotu, klient (użytkownik) tworzy obiekt **SubmitItemCommand** i przekazuje go do systemu, który wywołuje **execute():**

**Diagram sekwencji:**

User -> SubmitItemCommand : execute()

SubmitItemCommand -> Item : submit()

# 2. Wzorzec: Observer

### 2.1 Motywacja

W aplikacji istnieje potrzeba informowania użytkowników o zdarzeniach, takich jak zmiana statusu produktu (np. został sprzedany, usunięty, zmieniono cenę). Zastosowanie wzorca **Observer** umożliwia obiektom (np. **User2**) subskrybowanie obiektów **Item** i automatyczne otrzymywanie powiadomień przy każdej zmianie.

**Konsekwencje zastosowania wzorca:**

* Luźnie powiązanie między obiektami.
* Skalowalna architektura powiadamiania.
* Możliwość wielokrotnego wykorzystania kodu.

### 2.2 Struktura przed zastosowaniem wzorca

Przed zastosowaniem **Item** nie miał zdolności informowania innych obiektów o zmianach:

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

### 2.3 Struktura po zastosowaniu wzorca

Dodano abstrakcyjne klasy **Observer** i **Subject**. Klasa **Item** dziedziczy po **Subject**, a **User2** po **Observer**:

Obraz zawierający tekst, zrzut ekranu, Czcionka, wyświetlacz

Zawartość wygenerowana przez AI może być niepoprawna.

Obraz zawierający tekst, linia, diagram, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

### 2.4 Współdziałanie po zastosowaniu wzorca

Gdy **Item** zmienia swój stan (np. zmienia cenę), wywołuje **notifyObservers()**, co skutkuje wywołaniem **update()** we wszystkich zarejestrowanych **User2**:

**Diagram sekwencji:**

Item -> Subject : notifyObservers()

loop for each observer

Subject -> User2 : update()

end

# Podsumowanie

Zastosowanie wzorców czynnościowych Command i Observer pozwoliło poprawić jakość architektury projektu poprzez:

* separację obowiązków,
* zwiększenie możliwości rozbudowy i testowania systemu,
* umożliwienie lepszej obsługi interakcji między użytkownikami a produktami.