1. Fabryka Abstrakcyjna (User Factory)

Tytuł

Wdrożenie wzorca Fabryka Abstrakcyjna dla hierarchii użytkowników

Motywacja

W systemie e-commerce istnieje potrzeba tworzenia różnych typów użytkowników (Admin, Buyer, Seller) o różnych uprawnieniach i funkcjonalnościach. Fabryka Abstrakcyjna pozwala na:

* Centralizację logiki tworzenia obiektów
* Uproszczenie dodawania nowych typów użytkowników
* Zapewnienie spójności tworzonych obiektów
* Ukrycie szczegółów implementacyjnych przed klientem

Konsekwencje zastosowania:

* Zwiększenie elastyczności systemu
* Łatwiejsze utrzymanie kodu
* Możliwość prostej rozbudowy hierarchii użytkowników

Diagram klas:

[UserFactory] <|.. [StandardUserFactory]

[User] <|-- [Admin]

[User] <|-- [Buyer]

[User] <|-- [Seller]

Współdziałanie elementów

Proces tworzenia użytkownika:

1. Klient żąda konkretnego typu użytkownika od fabryki
2. Fabryka tworzy odpowiedni obiekt
3. Zwraca gotowy obiekt użytkownika

Diagram sekwencji:

Klient -> StandardUserFactory: create\_admin(...)

StandardUserFactory -> Admin: new(...)

Admin --> StandardUserFactory: obiekt

StandardUserFactory --> Klient: obiekt

2. Budowniczy (Transaction Builder)

Tytuł

Wdrożenie wzorca Budowniczy dla procesu tworzenia transakcji

Motywacja

Transakcje w systemie e-commerce mogą mieć różne konfiguracje (status, daty, kupujący). Budowniczy pozwala na:

* Stopniowe konstruowanie złożonych obiektów
* Tworzenie obiektów z różnymi kombinacjami parametrów
* Zapewnienie spójności obiektów nawet przy pominięciu niektórych parametrów
* Poprawę czytelności kodu poprzez fluent interface

Konsekwencje zastosowania:

* Elastyczność w tworzeniu obiektów
* Łatwiejsze zarządzanie opcjonalnymi parametrami
* Możliwość tworzenia predefiniowanych konfiguracji (Director)

Stan przed wdrożeniem wzorca

Przed refaktoryzacją transakcje tworzono poprzez bezpośrednie wywołanie konstruktora z wieloma parametrami, co prowadziło do:

* Długich i trudnych do zarządzania list parametrów
* Trudności w obsłudze opcjonalnych pól
* Mało czytelnego kodu

Fragment kodu przed refaktoryzacją:

python

Copy

transaction = Transaction(1, buyer, "PENDING", datetime.now(), None)

Struktura po wdrożeniu wzorca

Po zastosowaniu wzorca wprowadzono:

1. Interfejs TransactionBuilder z metodami do ustawiania parametrów
2. Konkretną implementację StandardTransactionBuilder
3. Klasę TransactionDirector dla predefiniowanych konfiguracji
4. Główną klasę Transaction

Diagram klas:

Copy

[TransactionBuilder] <|.. [StandardTransactionBuilder]

[TransactionDirector] o-> [TransactionBuilder]

[StandardTransactionBuilder] --> [Transaction]

Współdziałanie elementów

Proces tworzenia transakcji:

1. Klient tworzy budowniczego
2. Ustawia wymagane parametry (metodą płynnego interfejsu)
3. Wywołuje metodę build() aby otrzymać gotowy obiekt
4. (Opcjonalnie) Może użyć Directora dla predefiniowanych konfiguracji

Diagram sekwencji:

Copy

Klient -> StandardTransactionBuilder: new()

Klient -> StandardTransactionBuilder: set\_id(1)

Klient -> StandardTransactionBuilder: set\_buyer(buyer)

Klient -> StandardTransactionBuilder: set\_status("PENDING")

Klient -> StandardTransactionBuilder: build()

StandardTransactionBuilder -> Transaction: new(...)

Transaction --> StandardTransactionBuilder: obiekt

StandardTransactionBuilder --> Klient: obiekt

Wnioski

1. **Fabryka Abstrakcyjna**:
   * Znacząco uprościła proces tworzenia różnych typów użytkowników
   * Umożliwiła centralne zarządzanie logiką tworzenia obiektów
   * Ułatwiła przyszłe rozszerzanie systemu o nowe typy użytkowników
2. **Budowniczy**:
   * Zapewnił elastyczność w tworzeniu obiektów Transaction
   * Poprawił czytelność kodu poprzez metodę płynnego interfejsu (fluent interface)
   * Umożliwił tworzenie obiektów z różnymi kombinacjami parametrów