Sprawozdanie: Wdrożenie Wzorców Kreacyjnych w Systemie E-commerce

1. Wstęp

W ramach projektu systemu e-commerce zaimplementowano dwa wzorce kreacyjne: **Fabrykę Abstrakcyjną** dla hierarchii użytkowników oraz **Budowniczego** dla procesu tworzenia transakcji. Poniższe sprawozdanie prezentuje szczegóły implementacji w Pythonie.

2. Implementacja w Pythonie

2.1. Fabryka Abstrakcyjna (User Factory), przykład:

from abc import ABC, abstractmethod

from datetime import datetime

# Hierarchia klas użytkowników

class User:

def \_\_init\_\_(self, name, surname, email, password):

self.name = name

self.surname = surname

self.email = email

self.password = password

class Admin(User):

def \_\_init\_\_(self, name, surname, email, password):

super().\_\_init\_\_(name, surname, email, password)

self.admin = True

def get\_user(self):

# Implementacja

pass

class Buyer(User):

def buy(self, transaction):

# Implementacja

pass

class Seller(User):

def sell(self, transaction):

# Implementacja

pass

# Interfejs Fabryki Abstrakcyjnej

class UserFactory(ABC):

@abstractmethod

def create\_user(self, name, surname, email, password):

pass

@abstractmethod

def create\_admin(self, name, surname, email, password):

pass

@abstractmethod

def create\_buyer(self, name, surname, email, password):

pass

@abstractmethod

def create\_seller(self, name, surname, email, password):

pass

# Konkretna implementacja fabryki

class StandardUserFactory(UserFactory):

def create\_user(self, name, surname, email, password):

return User(name, surname, email, password)

def create\_admin(self, name, surname, email, password):

return Admin(name, surname, email, password)

def create\_buyer(self, name, surname, email, password):

return Buyer(name, surname, email, password)

def create\_seller(self, name, surname, email, password):

return Seller(name, surname, email, password)

# Przykład użycia

factory = StandardUserFactory()

admin = factory.create\_admin("Jan", "Kowalski", "admin@example.com", "secure123")

buyer = factory.create\_buyer("Anna", "Nowak", "anna@example.com", "pass123")

2.2. Budowniczy (Transaction Builder)

class Transaction:

def \_\_init\_\_(self, id, buyer, status, created\_date, finished\_date=None):

self.id = id

self.buyer = buyer

self.status = status

self.created\_date = created\_date

self.finished\_date = finished\_date

def return\_transaction(self):

# Implementacja

pass

# Interfejs Budowniczego

class TransactionBuilder(ABC):

@abstractmethod

def set\_id(self, id):

pass

@abstractmethod

def set\_buyer(self, buyer):

pass

@abstractmethod

def set\_status(self, status):

pass

@abstractmethod

def set\_created\_date(self, date):

pass

@abstractmethod

def set\_finished\_date(self, date):

pass

@abstractmethod

def build(self):

pass

# Konkretna implementacja budowniczego

class StandardTransactionBuilder(TransactionBuilder):

def \_\_init\_\_(self):

self.id = None

self.buyer = None

self.status = "CREATED"

self.created\_date = datetime.now()

self.finished\_date = None

def set\_id(self, id):

self.id = id

return self

def set\_buyer(self, buyer):

self.buyer = buyer

return self

def set\_status(self, status):

self.status = status

return self

def set\_created\_date(self, date):

self.created\_date = date

return self

def set\_finished\_date(self, date):

self.finished\_date = date

return self

def build(self):

return Transaction(

self.id,

self.buyer,

self.status,

self.created\_date,

self.finished\_date

)

# Director dla bardziej złożonych przypadków

class TransactionDirector:

def \_\_init\_\_(self, builder):

self.builder = builder

def construct\_simple\_transaction(self, buyer):

return (self.builder

.set\_buyer(buyer)

.set\_status("CREATED")

.set\_created\_date(datetime.now())

.build())

def construct\_complete\_transaction(self, buyer, status):

return (self.builder

.set\_buyer(buyer)

.set\_status(status)

.set\_created\_date(datetime.now())

.set\_finished\_date(datetime.now())

.build())

# Przykład użycia

builder = StandardTransactionBuilder()

director = TransactionDirector(builder)

transaction1 = (builder

.set\_id(1)

.set\_buyer(buyer)

.set\_status("PENDING")

.build())

transaction2 = director.construct\_simple\_transaction(buyer)

5. 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