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