Creationals Patterns

🧪 Singleton

- **S** – często łamie SRP, bo odpowiada i za logikę, i za kontrolę instancji.
- **L** – narusza zasadę otwarte-zamknięte przy potrzebie rozszerzeń.
- Może być użyteczny, ale trzeba go stosować ostrożnie – często niezgodny z SOLID.

🧪 Fabryka (Factory Method)

- **O** – nowe typy produktów można dodawać bez modyfikowania fabryki.
- **D** – klient zależy od abstrakcji, a nie konkretnych klas.

🏗️ Budowniczy (Builder)

- **S** – oddziela tworzenie złożonego obiektu od jego reprezentacji.
- **O** – można rozszerzyć o nowe typy konstrukcji bez zmian klienta.
- **I** – klient używa tylko tych metod budowniczego, które potrzebuje.

♻️ Prototyp (Prototype)

- **S** – każda klasa prototypu odpowiada za kopiowanie samej siebie.
- **O** – można rozszerzyć zachowanie klonowania bez zmiany klienta.
- **L** – subtelnie, bo klient powinien móc korzystać z dowolnego prototypu w taki sam sposób.

🧪 Abstrakcyjna fabryka (Abstract Factory)

- **O** – możesz dodawać nowe typy produktów, nie zmieniając klienta.

- **D** – klient zależy tylko od interfejsu fabryki.

Structurals Patterns

🧩 Adapter (Adapter)

- **S** – adapter odpowiada tylko za translację interfejsów.
- **O** – możesz wprowadzać nowe adaptery bez modyfikacji oryginalnych klas.
- **D** – klient zależy od interfejsu, a nie od konkretnej implementacji.

🧱 Dekorator (Decorator)

- **O** – możesz dodawać nowe funkcje bez modyfikowania istniejących klas.
- **S** – dekorator odpowiada za jedną rozszerzoną funkcję.
- **D** – dekorator używa interfejsu bazowego, nie konkretnej klasy.

🔐 Pełnomocnik (Proxy)

- **S** – pełnomocnik odpowiada za kontrolę dostępu, cache lub opóźnione ładowanie.
- **O** – dodajesz nowe rodzaje proxy bez zmian w kliencie.
- **D** – klient zależy tylko od interfejsu (np. `IService`), nie od konkretnej implementacji.

📦 Kompozyt (Composite)

- **S** – każdy komponent (liść lub kompozyt) ma jedną odpowiedzialność.
- **O** – można dodawać nowe typy komponentów bez modyfikacji struktury.
- **L** – klient może używać obiektu kompozytu i liścia zamiennie.

🩹 Fasada (Facade)

- **S** – fasada upraszcza interakcję z złożonym systemem.
- **O** – możesz dodać nowe systemy wewnętrzne bez zmiany fasady.
- **D** – klient zależy od uproszczonego interfejsu, nie od detali.

🗂️ Most (Bridge)

- **S** – abstrakcja i implementacja mają oddzielne odpowiedzialności.
- **O** – możesz zmieniać implementację niezależnie od abstrakcji.
- **D** – abstrakcja zależy od interfejsu implementacji.

Behaviorals Patterns

📚 Szablon metody (Template Method)

- **S** – szablon definiuje ogólny algorytm, a podklasy implementują szczegóły.
- **O** – można zmieniać kroki algorytmu, dodając nowe podklasy.
- **L** – klasy potomne powinny zachować ogólne założenia działania metody bazowej.

🔁 Strategia (Strategy)

- **S** – każda strategia ma pojedynczą odpowiedzialność.
- **O** – możesz dodać nowe strategie bez zmian w kontekście.
- **D** – kontekst zależy od abstrakcji (interfejsu strategii), nie od konkretów.

🔁 Obserwator (Observer)

- **O** – można dodawać nowe typy obserwatorów bez zmian w obiekcie obserwowanym.
- **S** – każdy obserwator i obiekt obserwowany mają jedną odpowiedzialność.

- **D** – obserwatorzy zależą od interfejsu, nie od konkretnego nadawcy.

🔀 Łańcuch zobowiązań (Chain of Responsibility)

- **S** – każdy handler ma swoją odpowiedzialność.
- **O** – można dodać nowe ogniwa łańcucha bez zmian pozostałych.
- **D** – zależność od interfejsu handlera, nie konkretnych klas.

🧠 Interpreter

- **S** – każda reguła/wyrażenie reprezentuje pojedynczą logikę.
- **O** – można dodawać nowe interpretacje bez zmiany parsera.
- **D** – parser zależy od abstrakcji `IExpression`.

🎮 Polecenie (Command)

- **S** – obiekt polecenia reprezentuje jedną operację.
- **O** – nowe polecenia można dodawać bez modyfikacji wykonawcy.
- **D** – klient zależy od interfejsu `ICommand`.

⏮️ Memento (Pamiątka)

- **S** – pamiątka przechowuje tylko stan, nie logikę.
- **O** – logikę przywracania stanu można rozszerzać bez zmiany klienta.
- **I** – klient ma dostęp tylko do tych metod pamiątki, które są mu potrzebne.

💾 Stan (State)

- **S** – każdy stan odpowiada za jedno zachowanie.
- **O** – można dodawać nowe stany bez modyfikacji kontekstu.
- **D** – kontekst zależy od interfejsu stanu, nie od konkretnych implementacji.

🧭 Mediator (Mediator)

- **S** – mediator centralizuje komunikację między obiektami.
- **O** – nowe interakcje można dodać przez mediator, bez zmian w komponentach.
- **D** – komponenty zależą od abstrakcyjnego mediatora, nie od siebie nawzajem.

🧮 Interpreter

- **S** – każda reguła odpowiada za interpretację jednego wyrażenia.
- **O** – można dodać nowe typy wyrażeń bez zmiany parsera.
- **D** – parser korzysta z interfejsu `IExpression`.

🧠 Wizytator (Visitor)

- **S** – wizytator oddziela logikę działania od struktury danych.
- **O** – można dodawać nowe operacje bez zmian w strukturze danych.
- **I** – elementy struktury nie muszą znać wszystkich metod wizytatora – choć to czasem bywa łamane.

🔁 Iterator

- **S** – iterator odpowiada tylko za dostęp do elementów kolekcji.
- **O** – można dodać nowe sposoby iterowania bez modyfikowania kolekcji.
- **I** – użytkownik korzysta tylko z interfejsu iteratora (`IEnumerator` itp.).

🧰 Ładowanie opóźnione (Lazy Initialization / Virtual Proxy)

- **S** – odpowiedzialność za inicjalizację obiektu jest oddzielona od jego użytkowania.
- **O** – można zmienić sposób inicjalizacji bez zmiany klienta.

- **D** – klient korzysta z abstrakcji (np. `IDataProvider`), nie znając szczegółów inicjalizacji.