WZORCE PROJEKTOWE GOF

WZORCE PROJEKTOWE

- Wzorce projektowe to zestawy rozwiązań często występujących problemów. Są one szablonem, który przedstawia abstrakcyjne rozwiązanie. Wybrane rozwiązanie dostosowuje się do danego problemu dla konkretnego języka programowania.
- Wzorzec projektowy umożliwia projektowanie klas i obiektów wielokrotnego użytku, a także określenie relacji między obiektami i klasami.

- **Budowa wzorca:**
  - **Cel** pobieżnie opisuje problem i rozwiązanie.
  - **Motywacja** rozszerza opis problemu i rozwiązania jakie umożliwia dany wzorzec.
  - **Struktura** ukazuje poszczególne części wzorca i powiązania między nimi.
  - **Przykład kodu** w wybranym języku programowania pomaga zrozumieć ideę wzorca.

WZORCE PROJEKTOWE HISTORIA

- Motyw wzorców projektowych zyskał na sile w 1994 roku dzięki Gang of Four (założonej przez Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides). Napisali oni książkę "Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku", w której opisali 23 wzorce projektowe, które znane są jako wzorce projektowe GoF.
- Wzorce projektowe GoF są zazwyczaj opisywane za pomocą notacji graficznej, takiej jak diagram przypadków użycia. Notacja opisuje klasy i obiekty, a także relacje między nimi.

WZORCE PROJEKTOWE PODZIAŁ

Wzorce kreacyjne

- Przedstawiają mechanizmy tworzenia obiektów, zwiększających elastyczność i ułatwiających ponowne użycie kodu

Wzorce strukturalne

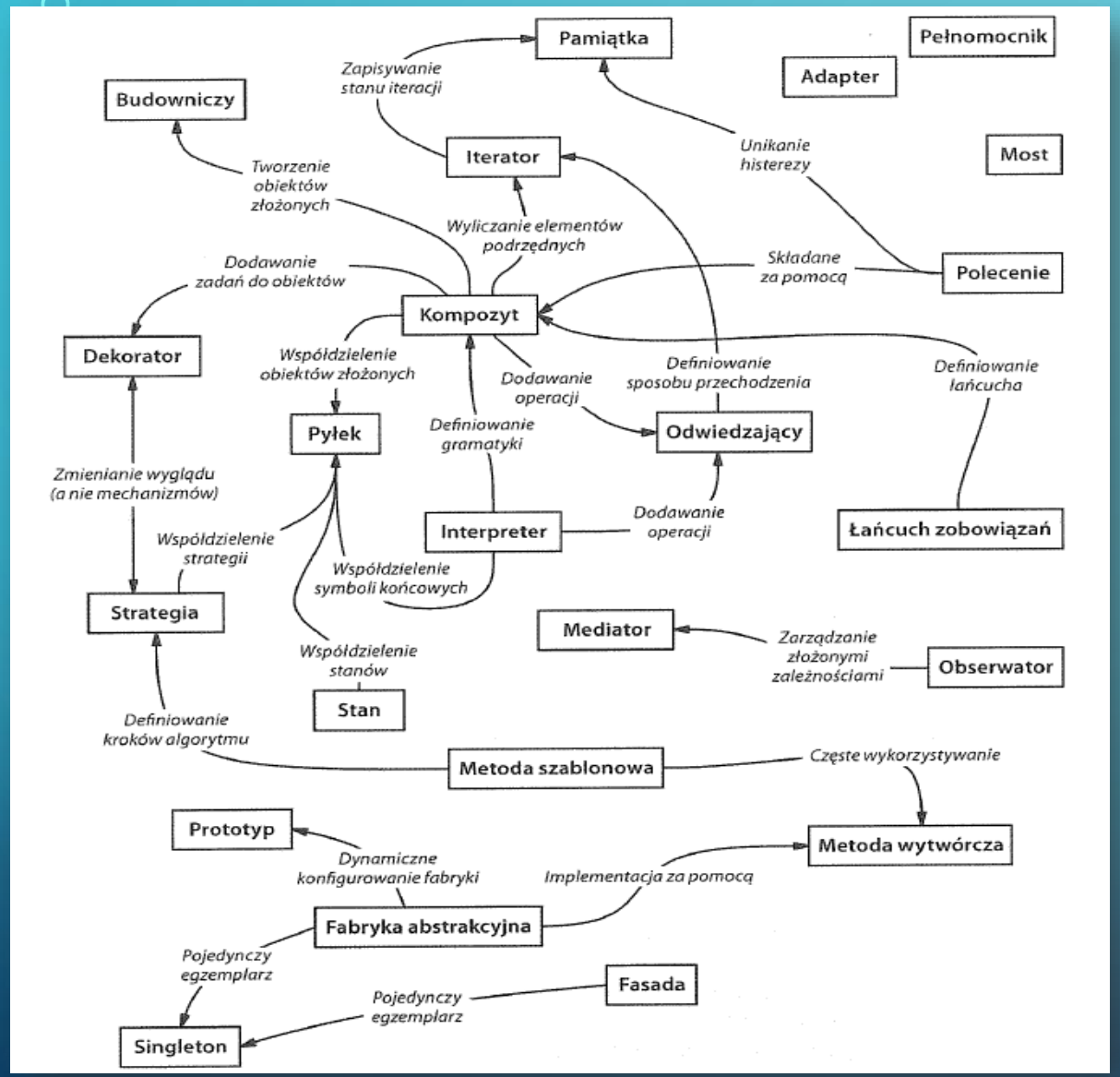
- Wyjaśniają sposób w jaki można składać obiekty i klasy w większe struktury, zachowując przy okazji elastyczność i efektywność.

Wzorce behawioralne

- Dotyczą algorytmów i podziału zadań pomiędzy obiektami.

		Przeznaczenie		
		Kreacyjne	**Strukturalne**	**Behawioralne**
Zakres	**Klasa**	FactoryMethod	Adapter	Interpreter Template Method
	Obiekt	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Proxy Flyweight	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

PRZYKŁAD RELACJI JAKIE ZACHODZĄ MIĘDZY WZORCAMI PROJEKTOWYMI



OGÓLNA SKŁADNIA WZORCÓW PROJEKTOWYCH

1. Nazwa wzorca

Opis powinien składać się z jednego lub dwóch słów, które mają jak najdokładniej opisywać dany problem, wzorzec.

2. Opis problemu

Powinno być zawarte kiedy należy stosować dany wzorzec, opisy specyficznych problemów projektowych. Może również być zawarta lista warunków które muszą być spełnione.

3. Rozwiązanie

Opis elementów które są w projekcie. Nie jest to gotowy przepis jakie ma być końcowe rozwiązanie, wzorzec zawsze przypomina szablon. Możemy się spotkać z listą warunków które muszą być spełnione

4. Konsekwencje

Koszty oraz zyski, wynikające bezpośrednio z zastosowania wzorca. Zazwyczaj w opisach konsekwencje nie są jawne, maja kluczowe znaczenie przy ocenie korzyści zastosowania danego wzorca projektowego

DOBÓR ODPOWIEDNIEGO WZORCA

Tutaj jest bardzo dużo zależności, od wybrania odpowiedniego wzorca zależy wiele parametrów

- Znalezienie odpowiedniego obiektu (przechowywanie danych jak i operują na nich procedury)
- Określenie szczegółowości obiektu (obiekty mogą przyjmować bardzo rozmaite wielkości, ilości. Mogą reprezentować wszystko)
- Określenie Interfejsów i Obiektów (każdy obiekt składa się na sygnaturę, zestaw wszystkich sygnatur to interfejs)
- Określenie Implementacji obiektów (implementacje obiektów wyznacza jego klasa)
- Dziedziczenie klas i interfejsów

FABRYKA ABSTRAKCYJNA

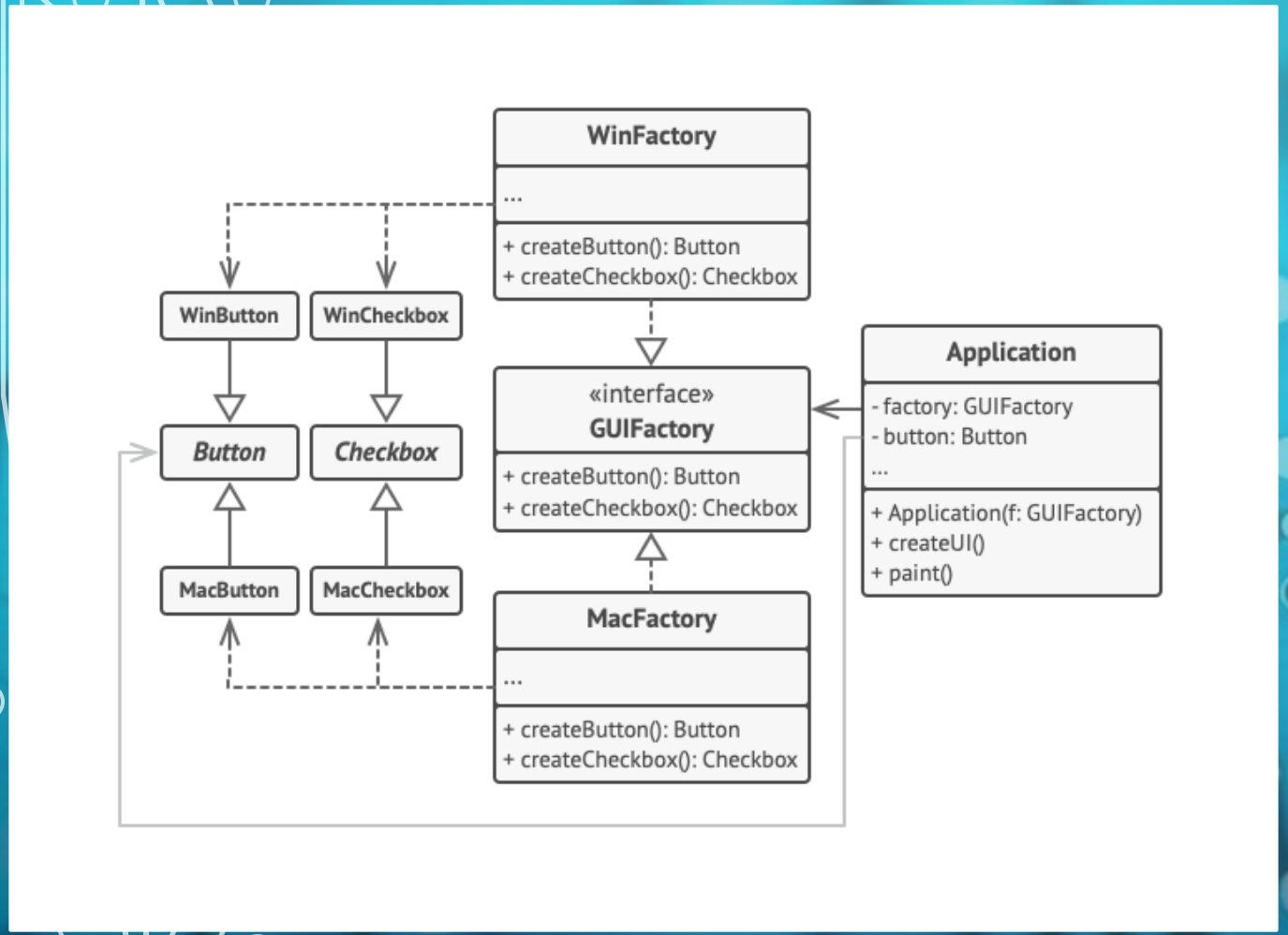
Czasem w projekcie zachodzi potrzeba stworzenia rodziny obiektów, które ściśle współpracują ze sobą. Przykładowo program działający na danej platformie do poprawnego działania wymaga obiektów, które są dedykowane dla niej. Do działania aplikacji na innej platformie jest potrzebny inny zestaw obiektów. Inicjowanie każdego obiektu osobno nie jest praktyczne.

Fabryka abstrakcyjna, to wzorzec kreacyjny, który idealnie nadaje się do wykorzystania w powyższym przykładzie. Dostarcza interfejs umożliwiający tworzenie obiektów z tej samej grupy(rodziny).

Zalety:

- Pewność, że obiekty stworzone przez fabrykę są ze sobą kompatybilne
- Zebranie kodu kreacyjnego obiektów w jednym miejscu(zasada pojedynczej odpowiedzialności)
- Wprowadzenie wsparcia dla nowych wariantów obiektów bez psucia istniejącego kodu klienckiego

FABRYKA ABSTRAKCYJNA - PRZYKŁAD



Przykład po lewej ilustruje zastosowanie fabryki abstrakcyjnej do tworzenia elementów interfejsu graficznego w zależności od systemu operacyjnego, na którym jest uruchamiana aplikacja. WinFactory jak i MacFactory implementuje interfejs GUIFactory. Zarówno jedna jak i druga tworzą obiekty typu Button i Checkbox natomiast każda z nich te obiekty wytwarza pod wybrany system. Dzięki temu kod kliencki nie jest sprzęgnięty z konkretnym typem buttona lub checkboxa.

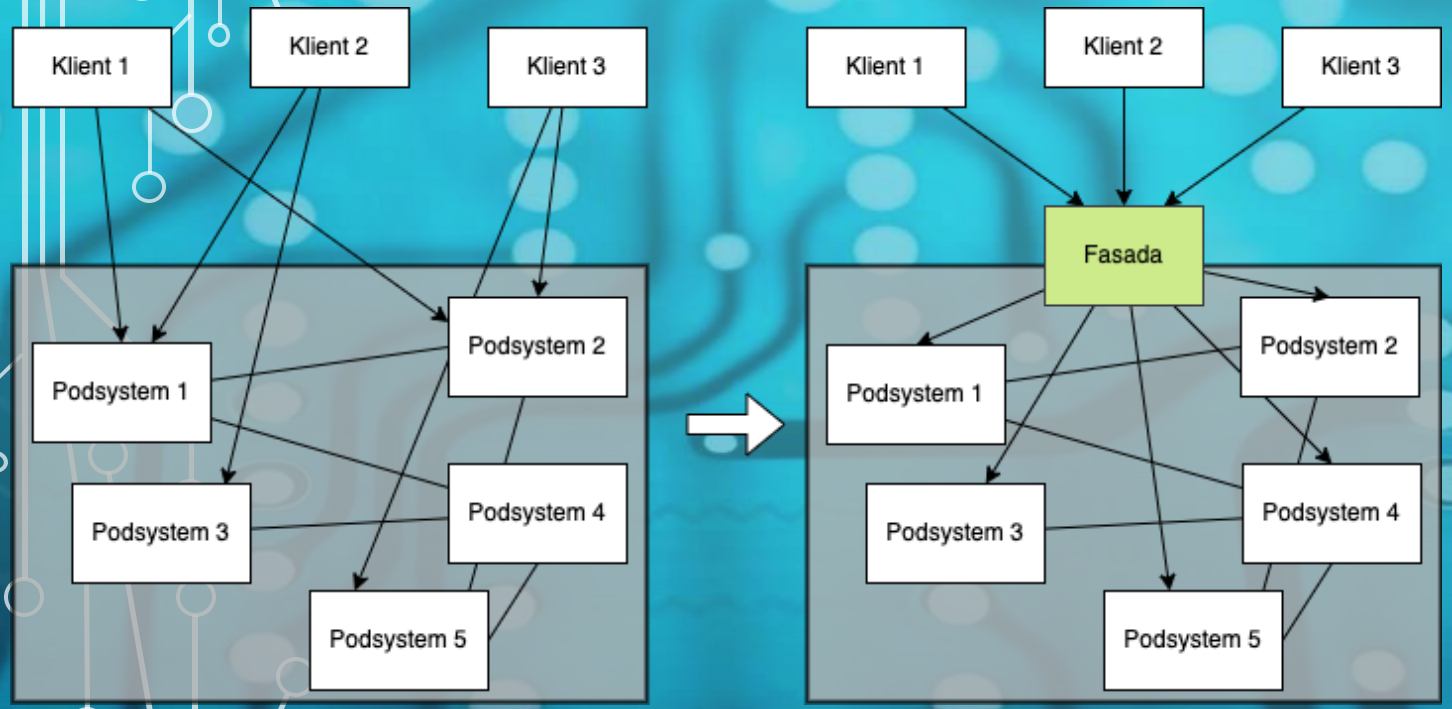
FASADA

Kiedy projekty stają się zbyt duże i skomplikowane, zazwyczaj dobrym rozwiązaniem jest zastosowanie zasady „dziel i rządź” i podzielenie systemu na mniej złożone podsystemy. Wówczas jednak klient takiego systemu musi samodzielnie odwoływać się do każdego podsystemu, przechowywać ich obiekty i dostosowywać swój kod, kiedy któryś z podsystemów zostanie zmodyfikowany. Aby tego uniknąć stosuje się wzorzec projektowy Fasada.

Czym jest Fasada?

- Wzorzec strukturalny
- Określa jednolity interfejs wyższego poziomu dla zbioru interfejsów z podsystemu, dzięki czemu ukrywa złożoność systemu
- Oddziela aplikacje klientów od komponentów podsystemu, co ułatwia korzystanie z podsystemu i zwiększa jego niezależność

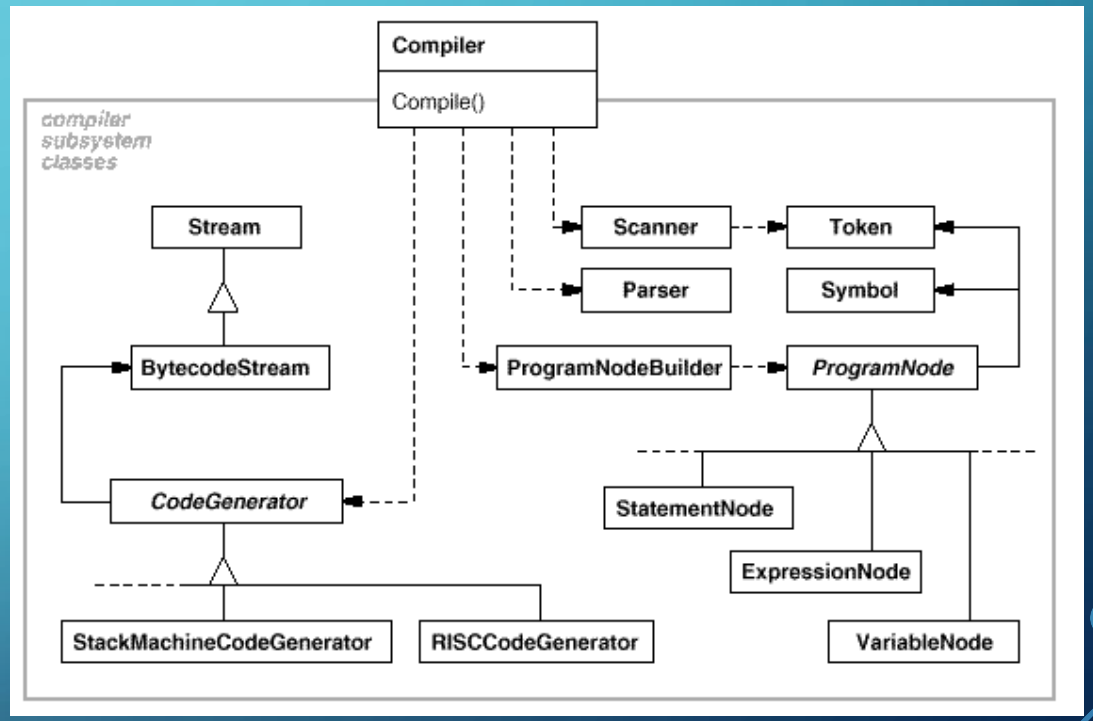
FASADA - ZASTOSOWANIE



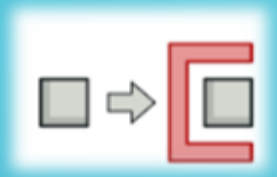
- Po podzieleniu systemu na mniejsze podsystemy każdy klient musi osobno komunikować się z odpowiednimi podsystemami.
- Po zastosowaniu wzorca projektowego Fasada klient może komunikować się z systemem poprzez fasadę, która przekształca żądania wysyłane do jej interfejsu i implementuje odwołania do odpowiednich podsystemów. Modyfikacja któregoś z podsystemów nie wpływa bezpośrednio na aplikacje klientów.
- Prosty domyślny interfejs, zazwyczaj odpowiedni dla większości aplikacji klientów. Jeśli jednak klient chce mieć większe możliwości samodzielnego dostosowania implementacji, to ciągle może komunikować się bezpośrednio z podsystemami.

FASADA - PRZYKŁAD

Jako przykład posłuży system do kompilowania aplikacji. Dla większości klientów proces kompilacji przebiega w ten sam sposób i nie chcą oni wdawać się w szczegóły działania tego systemu. Z tego powodu dodano Compiler, który w tym przykładzie odgrywa rolę fasady. Na żądanie klienta wykonuje odpowiednie odwołania do podsystemów, aby przeprowadzić domyślny proces kompilacji. Podsystemy zostały częściowo ukryte, a klientom udostępniono łatwiejszy w użyciu wysokopoziomowy interfejs. W przypadku bardziej wyspecjalizowanych aplikacji klient ma możliwość ominięcia fasady i odwoływania się bezpośrednio do podsystemów.



PROXY



Tworzenie skomplikowanych obiektów często wykorzystuje dużą część zasobów systemowych. W przypadku, gdy obiekt ten nie jest potrzebny przez cały czas działania aplikacji lub gdy posiadając kolekcję takich obiektów wykorzystujemy tylko część z nich, zasoby te są marnowane. W takiej sytuacji z pomocą przychodzi wzorzec Proxy.

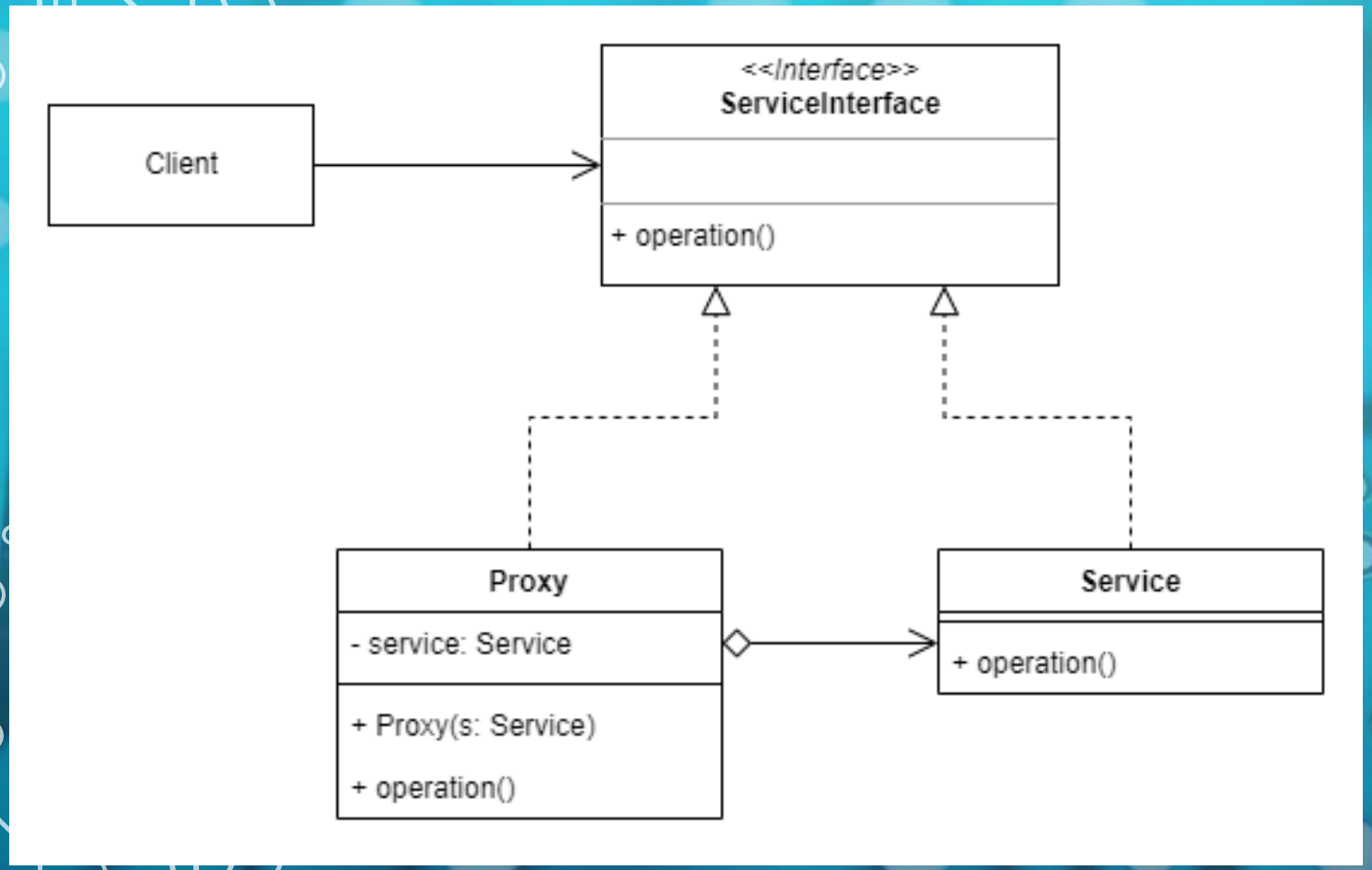
Proxy jest strukturalnym wzorcem projektowym. Pozwala stworzyć obiekt pełniący funkcję pełnomocnika innego obiektu, umożliwiając kontrolowany do niego dostęp.

PROXY – PRZYPADKI UŻYCIA

Istnieją cztery główne przypadki użycia wzorca Proxy:

1. **Virtual Proxy –** kontrola procesu tworzenia obiektu. Umożliwia jego inicjalizację na żądanie, dopiero w chwili, gdy obiekt ten jest potrzebny.
2. **Remote Proxy** - korzystanie z lokalnej implementacji obiektu znajdującego się w innej przestrzeni adresowej.
3. **Protective Proxy** - kontrola dostępu do obiektu.
4. **Smart Proxy** - wykonywanie dodatkowych akcji na obiekcie (np. zliczanie referencji)

PROXY - IMPLEMENTACJA



ServiceInterface jest interfejsem, który implementują dwie klasy: *Proxy* i *Service*. *Client*, korzystając ze wspólnego interfejsu, wywołuje metodę *operation()*, co docelowo ma wywołać metodę z klasy Service. Na drodze do jej wywołania stoi jednak pośrednik, czyli *Proxy*. Posiada on referencję do obiektu klasy *Service*. W swojej implementacji metody *operation()* wykonuje dodatkowe czynności, po czym wywołuję metodę docelową.

ITERATOR

- Wzorzec Iteratora stosowany jest w przypadku korzystania z kolekcji - popularnych typów danych agregujących grupy obiektów.
- Dla typów kolekcji, takich jak listy, sekwencyjne przechodzenie po elementach jest logiczne i związane z samym typem kolekcji, lecz w przypadku bardziej skomplikowanych struktur, takich jak drzewa, możliwości przechodzenia po kolejnych elementach jest wiele.
- Dodając kolejne algorytmy iterowania elementów kolekcji jej podstawowe zadanie polegające na efektywnym przechowywaniu danych jest przyćmiewane.
- Z pomocą przychodzi Iterator – behawioralny wzorzec projektowy, którego głównym celem jest wydzielenie zadań powiązanych z przechodzeniem przez elementy kolekcji. Pozwala on zapewnić nawigację po kolekcji bez eksponowania jej wewnętrznej struktury.

ITERATOR - ZASTOSOWANIE

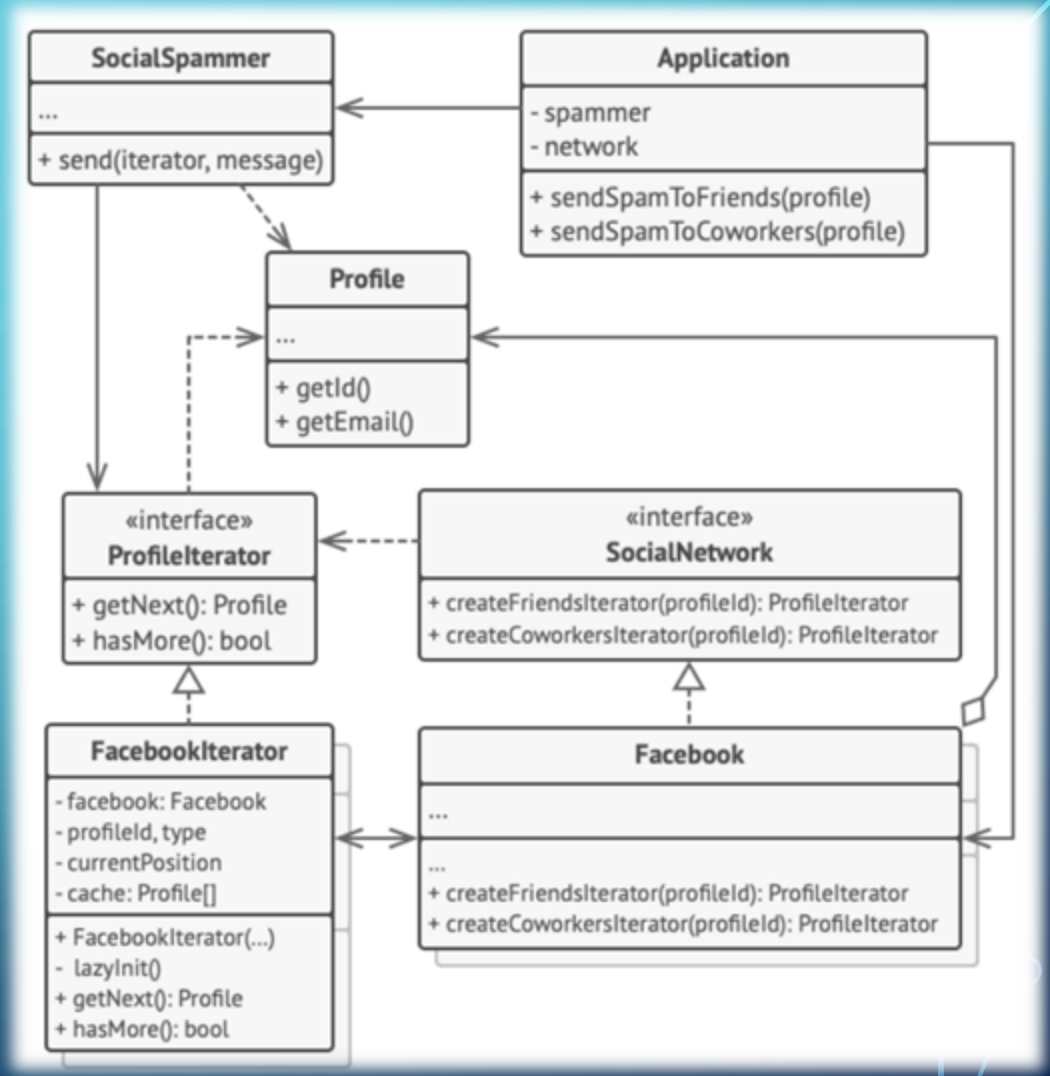
- Iterator ukrywa szczegóły implementacji złożonych struktur danych, udostępniając użytkownikowi kilka prostych metod dotyczących dostępu do kolejnych elementów kolekcji. Pozwala to zwiększyć bezpieczeństwo kolekcji przed niechcianymi zmianami, oraz wygodę korzystania z niej.
- Każdy obiekt iteratora przechowuje informację o swoim stanie, dzięki temu ta sama kolekcja może być przeglądana równolegle za pomocą kilku różnych iteratorów.
- Dzięki bazowaniu na interfejsach kod korzystający z Iteratora może w przyszłości zostać rozbudowany o współpracę z innymi kolekcjami. Wystarczy stworzyć implementację Iteratora oraz kolekcji działającą z innym typem danych.

ITERATOR - PRZYKŁAD

Na załączonym przykładzie, wzorzec Iteratora został wykorzystany w postaci interfejsu *ProfileIterator*, odpowiedzialnego za dostęp do kolejnych profili na portalu Facebook. Profile te zawarte są w obiektach implementujących interfejs *SocialNetwork*. Interfejs SocialNetwork deklaruje dwie metody służące do tworzenia odpowiednich iteratorów.

Przykładowymi implementacjami *ProfileIteratora* są:
- *FriendsIterator* - pozwalający na iterowanie po znajomych danego profilu.

- *ColleaguesIterator* - działający podobnie jak *FriendsIterator* z tą różnicą, że zwraca on tylko osoby pracujące w tej samej firmie.



WZORCE PROJEKTOWE A WZORCE TWORZENIA APLIKACJI BIZNESOWYCH

- Wzorce tworzenia aplikacji biznesowych (enterprise patterns) to zbiór rozwiązań dla powszechnych problemów występujących w oprogramowaniu przedsiębiorstw, spowodowanych ich złożonością
- Wzorce projektowe powstały jako pierwsze i bazują na nich inne wzorce, m. in. wzorce tworzenia aplikacji biznesowych
- Główną różnicą między tymi wzorcami jest ich cel – wzorce projektowe uporządkowują i optymalizują klasy oraz relacje między obiektami, zaś wzorce enterprise skupiają się na poprawie wykorzystania narzędzi i komunikacji między komponentami Javy EE
- Zastosowanie wzorców projektowych umożliwia ponowne wykorzystanie algorytmów i zwiększa elastyczność rozwiązania, zaś wzorce tworzenia aplikacji biznesowych ułatwiają korzystanie z narzędzi Javy EE i wprowadzanie zmian w architekturze

DZIĘKUJEMY ZA UWAGĘ!

- Julia Pabiańczyk
- Aleksander Dróżdż
- Natalia Bidzińska
- Rafał Czajka
- Dawid Dąbek
- Grzegorz Karkowski