Politechnika Wrocławska

Programowanie obiektowe

Wykład 12

Wzorce Projektowe

Prowadzący
dr inż. Paweł Maślak

Plan wykładu

- Czym są wzorce projektowe
- Rodzaje wzorców projektowych
- Wzorce kreacyjne
- Wzorce strukturalne
- Wzorce behawioralne

Czym jest wzorzec projektowy

- Wzorce projektowe to typowe rozwiązania problemów często napotykanych przy projektowaniu oprogramowania. Stanowią coś na kształt gotowych planów które można dostosować, aby rozwiązać powtarzający się problem w kodzie.

- Nie można jednak wybrać wzorca i po prostu skopiować go do programu, jak bibliotekę czy funkcję zewnętrznego dostawcy. Wzorzec nie jest konkretnym fragmentem kodu, ale ogólną koncepcją pozwalającą rozwiązać dany problem. Postępując według wzorca możesz zaimplementować rozwiązanie które będzie pasować do realiów twojego programu.

Czym jest wzorzec projektowy

- Wzorce często myli się z algorytmami, ponieważ obie koncepcje opisują typowe rozwiązanie jakiegoś znanego problemu. Algorytm jednak zawsze definiuje wyraźny zestaw czynności które prowadzą do celu, zaś wzorzec to wysokopoziomowy opis rozwiązania. Kod powstały na podstawie jednego wzorca może wyglądać zupełnie inaczej w różnych programach.

- Algorytm jest jak przepis kulinarny: oba mają wyraźnie określone etapy które trzeba wykonać w określonej kolejności by osiągnąć cel. Wzorzec bardziej przypomina strategię: znany jest wynik i założenia, ale dokładna kolejność implementacji należy do ciebie.

Czym jest wzorzec projektowy

- Większość wzorców posiada formalny opis, dzięki czemu każdy może odtworzyć ich ideę w różnych kontekstach. Oto sekcje na które zwykle dzieli się opis wzorca:

- **Cel** pobieżnie opisuje zarówno problem, jak i rozwiązanie.
- **Motywacja** rozszerza opis problemu i rozwiązania jakie umożliwia dany wzorzec.
- **Struktura klas** ukazuje poszczególne części wzorca i jak są ze sobą powiązane.
- **Przykład kodu** w którymś z popularnych języków programowania pomaga zrozumieć ideę wzorca.

Niektóre katalogi wzorców wymieniają inne użyteczne szczegóły, jak typowe zastosowanie wzorca, etapy implementacji i powiązania z innymi wzorcami.

Klasyfikacja wzorców

- Wzorce projektowe różnią się między sobą złożonością, poziomem szczegółowości i skalą w jakiej da się je zaimplementować w projektowanym systemie. Podoba mi się analogia do budowy dróg: można uczynić skrzyżowanie bezpieczniejszym montując tylko sygnalizację świetlną, albo tworząc wielopoziomowe rozjazdy z podziemnymi przejściami dla pieszych.

Klasyfikacja wzorców

- Najbardziej podstawowe i niskopoziomowe wzorce są często nazywane idiomami. Zazwyczaj stanowią funkcjonalność jakiegoś języka programowania.

- Najbardziej uniwersalnymi i wysokopoziomowymi wzorcami są wzorce architektoniczne. Deweloperzy mogą implementować je w niemal każdym języku. W przeciwieństwie do innych wzorców, mogą służyć zaprojektowaniu architektury całej aplikacji.

Klasyfikacja wzorców

- Wszystkie wzorce można skategoryzować według ich celu, bądź przeznaczenia. Oto 3 kategorie wzorców:

- **Wzorce kreacyjne** wprowadzają elastyczniejsze mechanizmy tworzenia obiektów i pozwalają na ponowne wykorzystanie istniejącego kodu.

- **Wzorce strukturalne** wyjaśniają jak składać obiekty i klasy w większe struktury, zachowując przy tym elastyczność i efektywność struktur.

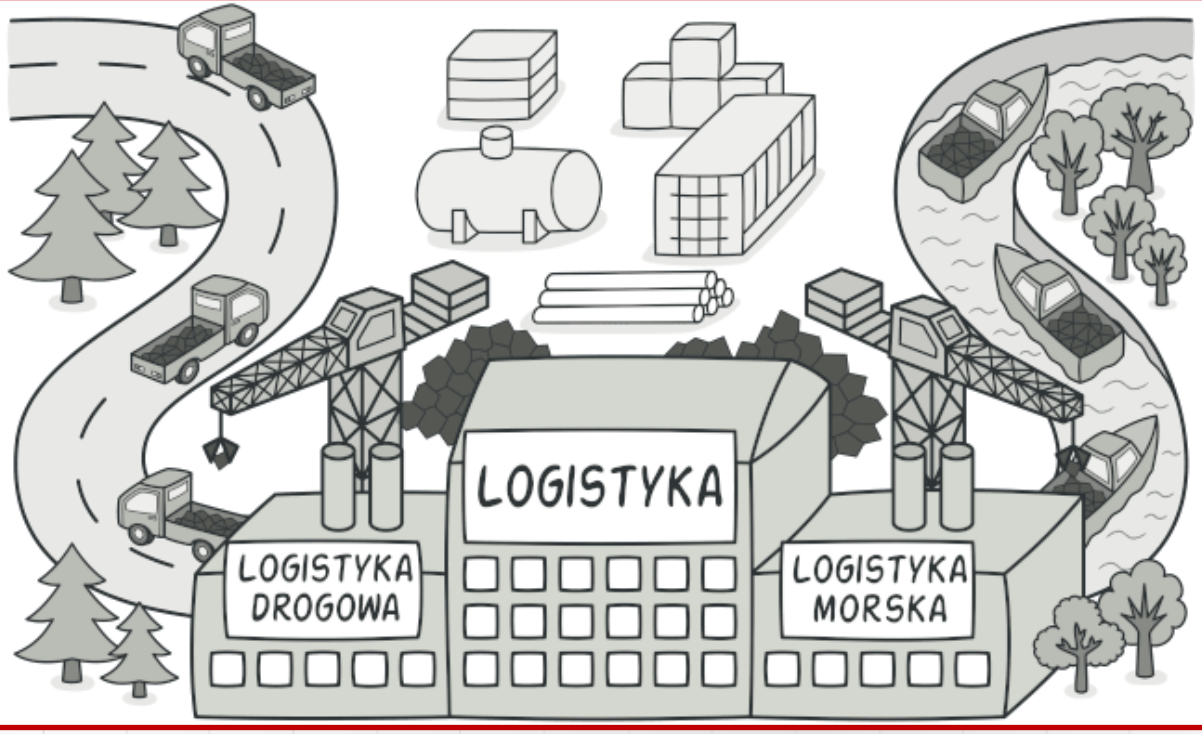
- **Wzorce behawioralne** które zajmują się efektywną komunikacją i podziałem obowiązków pomiędzy obiektami.

Wzorce kreacyjne

- Metoda wytwórcza
- Fabryka abstrakcyjna
- Budowniczy
- Prototyp
- Singleton

Wzorce kreacyjne

- **Cel: Metoda wytwórcza** jest kreacyjnym wzorcem projektowym, który udostępnia interfejs do tworzenia obiektów w ramach klasy bazowej, ale pozwala podklasom zmieniać typ tworzonych obiektów.



Wzorce kreacyjne

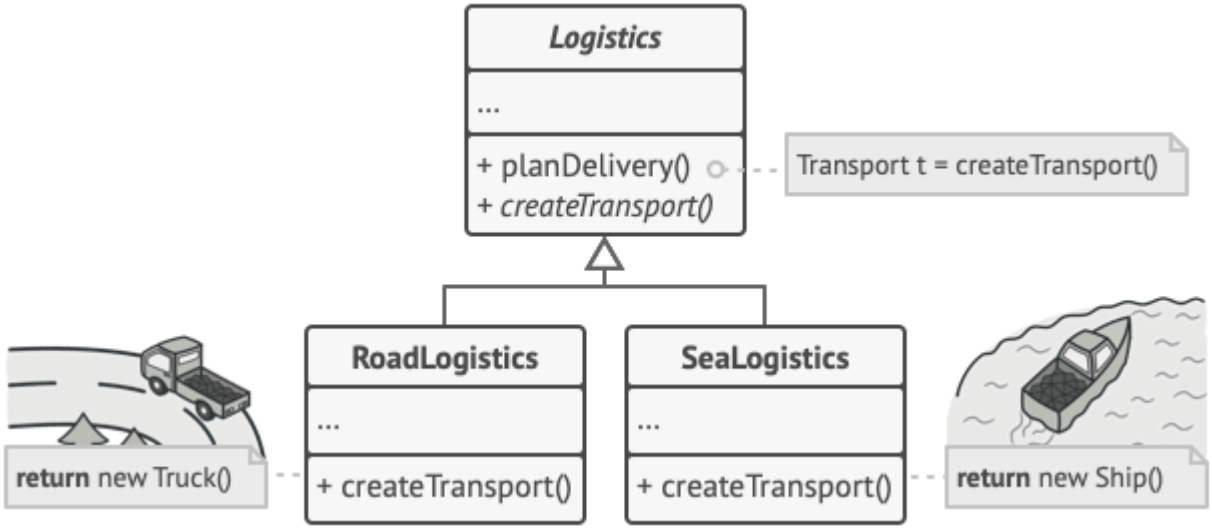
- **Problem:**
- Wyobraź sobie, że tworzysz aplikację do zarządzania logistyką. Pierwsza wersja twojej aplikacji pozwala jedynie na obsługę transportu za pośrednictwem ciężarówek, więc większość kodu znajduje się wewnątrz klasy Ciężarówka.

- Po jakimś czasie twoja aplikacja staje się całkiem popularna. Codziennie otrzymujesz tuzin próśb od firm realizujących spedycję morską, abyś dodał stosowną funkcjonalność do swej aplikacji.

- Świetna wiadomość, prawda? Ale co z kodem? W tej chwili większość twojego kodu jest powiązana z klasą Ciężarówka. Dodanie do aplikacji klasy Statki wymagałoby dokonania zmian w całym kodzie. Co więcej, jeśli później zdecydujesz się dodać kolejny rodzaj transportu, zapewne będziesz musiał dokonać tych zmian jeszcze jeden raz.
- Rezultatem powyższych działań będzie brzydki kod, pełen instrukcji warunkowych, których zadaniem będzie dostosowanie zachowania aplikacji zależnie od klasy transportu.

Wzorce kreacyjne

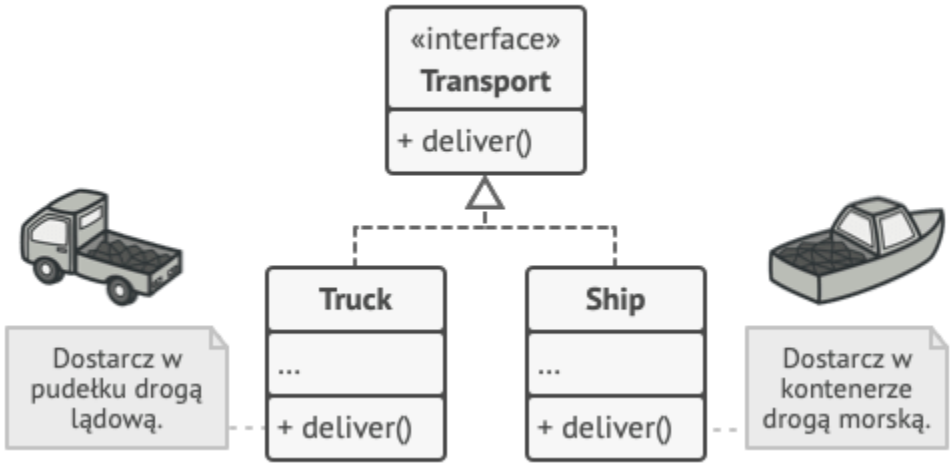
- **Rozwiazanie:**
- Wzorzec projektowy **Metody wytwórczej** proponuje zamianę bezpośrednich wywołań konstruktorów obiektów (wykorzystujących operator new) na wywołania specjalnej metody wytwórczej. Jednak nie przejmuj się tym: obiekty nadal powstają za pośrednictwem operatora new, ale teraz dokonuje się to za kulisami — z wnętrza metody wytwórczej. Obiekty zwracane przez metodę wytwórczą często są nazywane produktami.



Wzorce kreacyjne

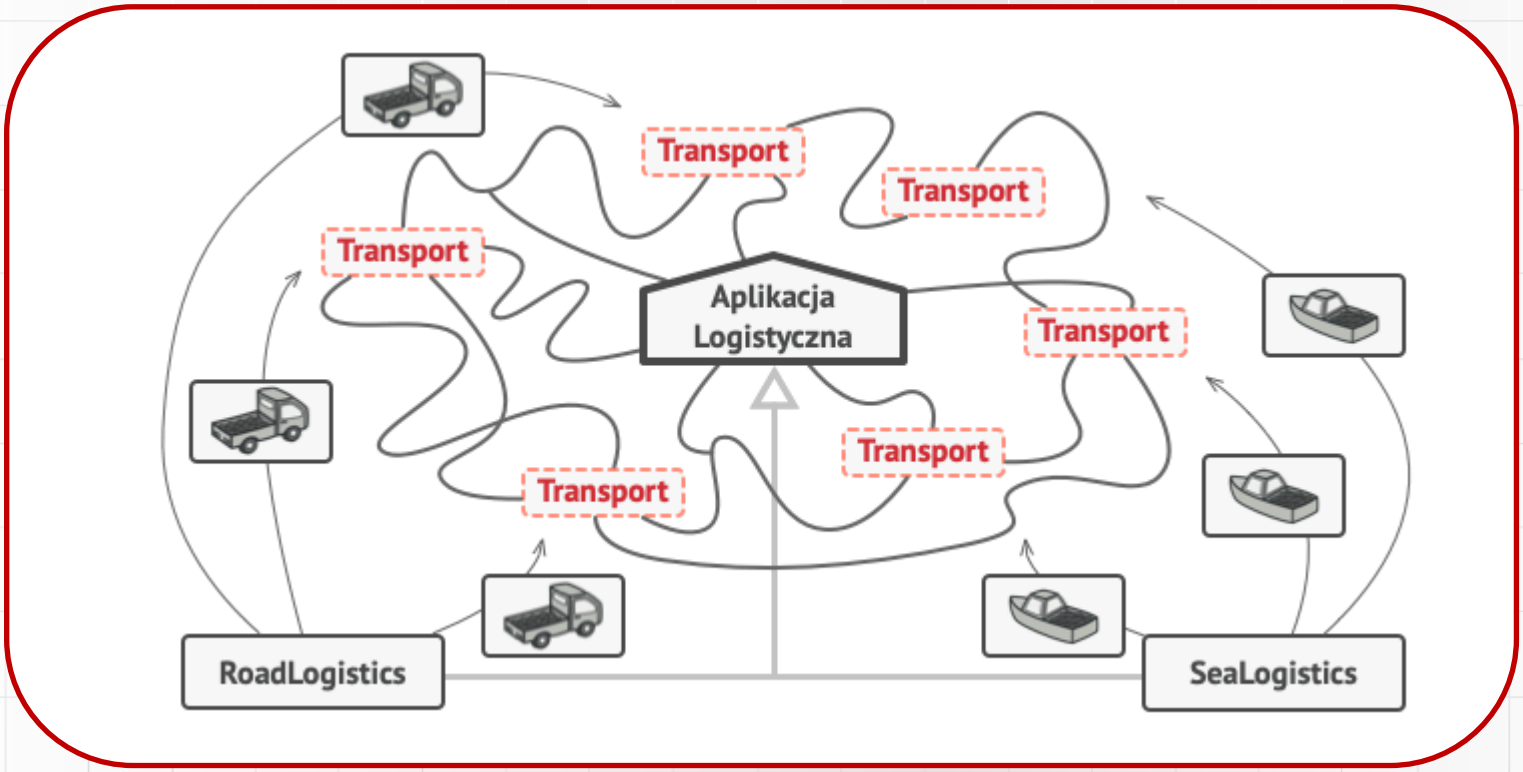
- Na pierwszy rzut oka zmiana ta może wydawać się bezcelowa. Przecież przenieśliśmy jedynie wywołanie konstruktora z jednej części programu do drugiej. Ale zwróć uwagę, że teraz możesz nadpisać metodę wytwórczą w podklasie, a tym samym zmienić klasę produktów zwracanych przez metodę.

- Istnieje jednak małe ograniczenie: podklasy mogą zwracać różne typy produktów tylko wtedy, gdy produkty te mają wspólną klasę bazową lub wspólny interfejs. Ponadto, zwracany typ metody wytwórczej w klasie bazowej powinien być zgodny z tym interfejsem.



Wzorce kreacyjne

- Na przykład zarówno klasy Ciężarówka, jak i Statek powinny implementować interfejs Transport, który z kolei deklaruje metodę dostarczaj. Każda klasa różnie implementuje tę metodę: ciężarówki dostarczają towar drogą lądową, statki drogą morską. Metoda wytwórcza znajdująca się w klasie LogistykaDrogowa zwraca obiekty Ciężarówka, zaś metoda wytwórcza w klasie LogistykaMorska zwraca Statki.



Wzorce kreacyjne

Zastosowanie:

- Stosuj Metodę Wytwórczą gdy nie wiesz z góry jakie typy obiektów pojawią się w twoim programie i jakie będą między nimi zależności.
- Korzystaj z Metody Wytwórczej gdy zamierzasz pozwolić użytkującym twą bibliotekę lub framework rozbudowywać jej wewnętrzne komponenty.
- Korzystaj z Metody wytwórczej gdy chcesz oszczędniej wykorzystać zasoby systemowe poprzez ponowne wykorzystanie już istniejących obiektów, zamiast odbudowywać je raz za razem.

Wzorce kreacyjne

Zalety

- Unikasz ścisłego sprzęgnięcia pomiędzy twórcą a konkretnymi produktami.
- Zasada pojedynczej odpowiedzialności. Możesz przenieść kod kreacyjny produktów w jedno miejsce programu, ułatwiając tym samym utrzymanie kodu.
- Zasada otwarte/zamknięte. Możesz wprowadzić do programu nowe typy produktów bez psucia istniejącego kodu klienckiego.

Wady

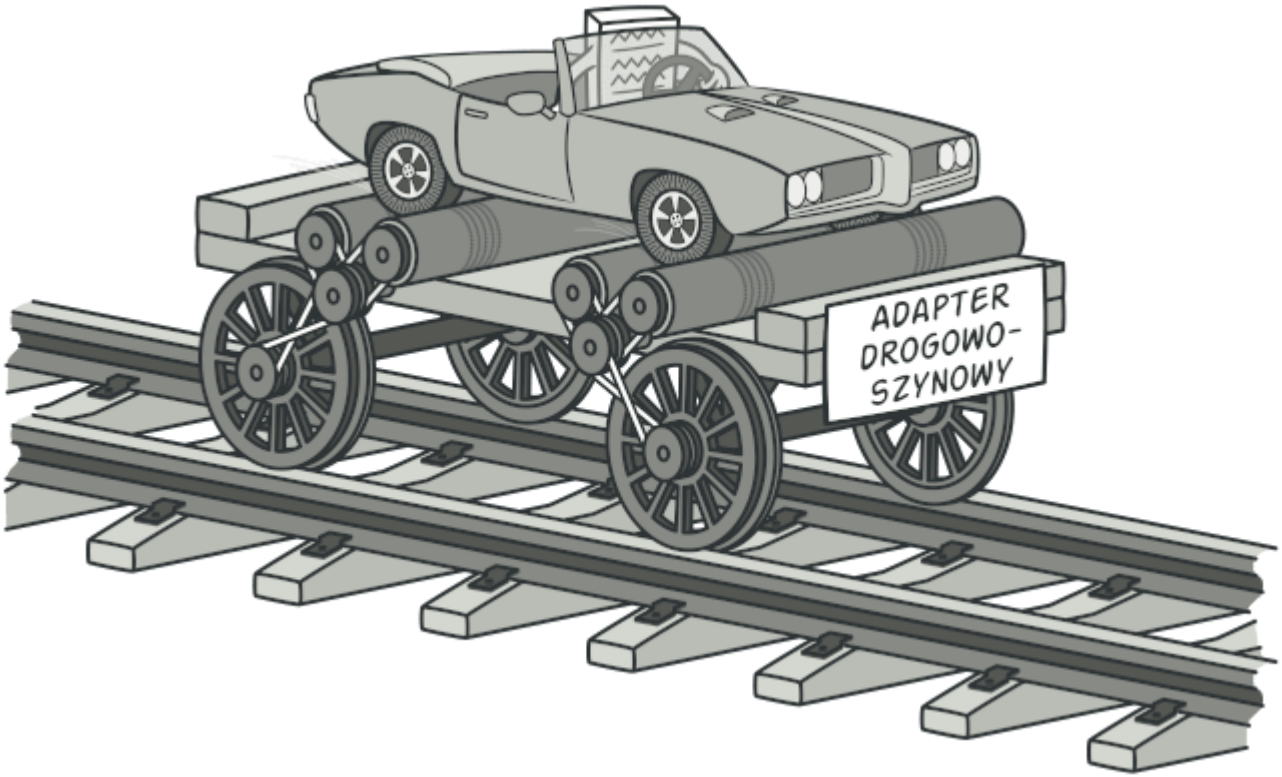
- Kod może się skomplikować, ponieważ aby zaimplementować wzorzec, musisz utworzyć liczne podklasy. W najlepszej sytuacji wprowadzisz ów wzorzec projektowy do już istniejącej hierarchii klas kreacyjnych.

Wzorce strukturalne

- Adapter
- Most
- Kompozyt
- Dekorator
- Fasada
- Pyłek
- Pełnomocnik

Wzorce strukturalne

- **Cel**: **Adapter** jest strukturalnym wzorcem projektowym pozwalającym na współdziałanie ze sobą obiektów o niekompatybilnych interfejsach.

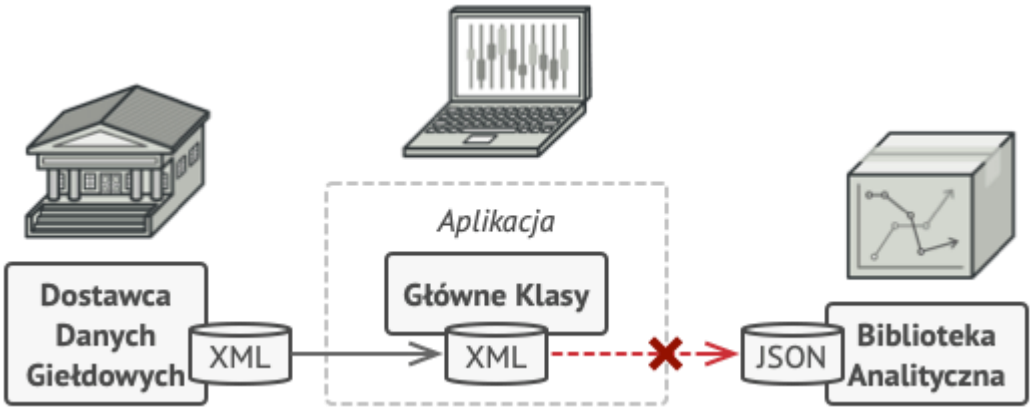


Wzorce strukturalne

Problem:

- Wyobraź sobie, że tworzysz aplikację monitorującą giełdę. Pobiera ona dane rynkowe z wielu źródeł w formacie XML, a następnie wyświetla ładnie wyglądające wykresy i diagramy.

- Na jakimś etapie postanawiasz wzbogacić aplikację poprzez dodanie inteligentnej biblioteki analitycznej innego producenta. Ale jest haczyk: biblioteka ta działa wyłącznie z danymi w formacie JSON.
- Można przerobić bibliotekę tak, aby obsługiwała XML. To jednak może naruszyć działanie istniejącego kodu korzystającego z tej biblioteki. Co gorsza, możesz nie mieć dostępu do kodu źródłowego biblioteki, co czyni ten plan niewykonalnym.



Wzorce strukturalne

Rozwiazanie:

- Możesz stworzyć adapter. Jest to specjalny obiekt konwertujący interfejs jednego z obiektów w taki sposób, że drugi obiekt go rozumie.

- Adapter stanowi swego rodzaju opakowanie dla obiektu, ukrywając szczegóły konwersji jakie odbywają się za kulisami. Obiekt opakowywany może nawet nie wiedzieć o istnieniu adaptera. Można na przykład opakować obiekt korzystający z jednostek kilometr i metr w adapter konwertujący te dane na jednostki imperialne, takie jak stopy i mile.

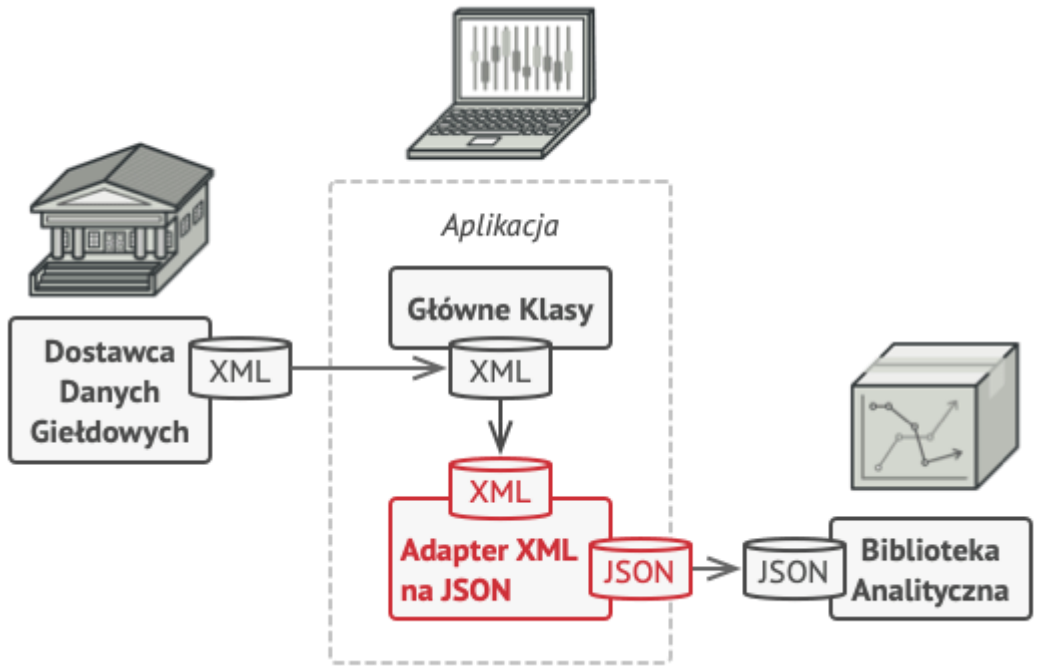
- Adaptery mogą nie tylko konwertować dane pomiędzy formatami, ale również pozwolić na współpracę obiektów o różnych interfejsach. Działa to tak:

1. Adapter uzyskuje interfejs kompatybilny z interfejsem jednego z obiektów.
2. Za pomocą tego interfejsu, istniejący obiekt może śmiało wywoływać metody adaptera.
3. Otrzymawszy wywołanie, adapter przekazuje je dalej, ale już w formacie obsługiwanym przez opakowany obiekt.

- Czasami jest nawet możliwe stworzenie adaptera dwukierunkowego, potrafiącego konwertować wywołania w obu kierunkach.

Wzorce strukturalne

Wróćmy do naszej aplikacji giełdowej. Aby rozwiązać dylemat niezgodności formatów, można stworzyć adaptery XML-do-JSON dla każdej klasy biblioteki analitycznej, która jest używana bezpośrednio przez nasz kod. Potem zaś możemy dostosować kod tak, aby komunikował się z biblioteką wyłącznie za pomocą adapterów. Gdy adapter otrzyma wywołanie, tłumaczy przychodzące dane XML na strukturę JSON i przekazuje wywołanie dalej, do odpowiedniej metody opakowywanego obiektu analitycznego.



Wzorce strukturalne

Analogia do prawdziwego życia:

- Gdy pierwszy raz podróżujesz z Polski do Wielkiej Brytanii, możesz zdziwić się próbując naładować laptop. Standardy wtyczek i gniazd są różne. Twoja wtyczka nie będzie pasować do brytyjskiego gniazdka. Problem można rozwiązać za pomocą przejściówki posiadającej gniazdo typu europejskiego oraz wtyk typu brytyjskiego.

Wzorce strukturalne

Zastosowanie:

- Stosuj klasę Adapter gdy chcesz wykorzystać jakąś istniejącą klasę, ale jej interfejs nie jest kompatybilny z resztą twojego programu.
- Stosuj ten wzorzec gdy chcesz wykorzystać ponownie wiele istniejących podklas którym brakuje jakiejś wspólnej funkcjonalności, niedającej się dodać do ich nadklasy.

Wzorce strukturalne

Zalety

- Zasada pojedynczej odpowiedzialności. Można oddzielić interfejs lub kod konwertujący dane od głównej logiki biznesowej programu.
- Zasada otwarte/zamknięte. Można wprowadzać do programu nowe typy adapterów bez psucia istniejącego kodu klienckiego, o ile będzie on korzystał z adapterów poprzez interfejs kliencki.

Wady

- Ogólna złożoność kodu zwiększa się, ponieważ trzeba wprowadzić zestaw nowych interfejsów i klas. Czasem łatwiej zmienić klasę udostępniającą jakąś potrzebną usługę, aby pasowała do reszty kodu.

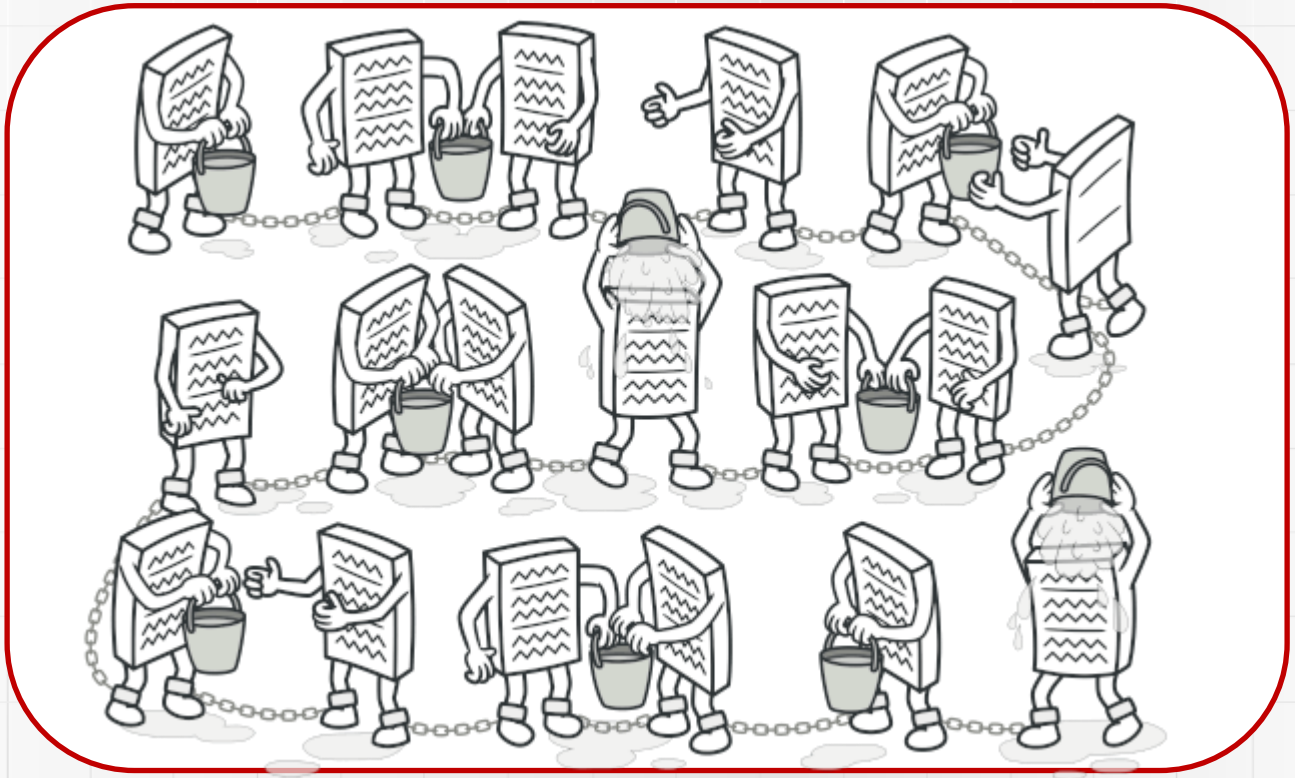
Wzorce behawioralne

- Łańcuch zobowiązań
- Polecenie
- Iterator
- Mediator
- Pamiątka
- Obserwator
- Stan
- Strategia
- Metoda Szablonowa
- Odwedzający

Wzorce behawioralne

Cel:

- **Łańcuch zobowiązań** jest behawioralnym wzorcem projektowym, który pozwala przekazywać żądania wzdłuż łańcucha obiektów obsługujących. Otrzymawszy żądanie, każdy z obiektów obsługujących decyduje o przetworzeniu żądania lub przekazaniu go do kolejnego obiektu obsługującego w łańcuchu.

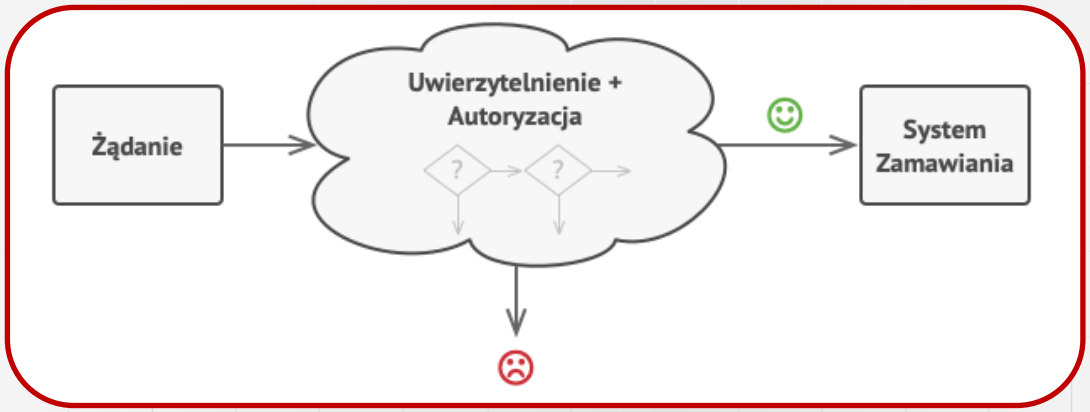


Wzorce behawioralne

Problem:

- Wyobraź sobie, że pracujesz nad systemem zamawiania online. Chcesz ograniczyć dostęp do systemu, by wyłącznie użytkownicy uwierzytelnieni mogli składać zamówienia. Ponadto użytkownicy z uprawnieniami administracyjnymi powinni mieć pełen dostęp do wszystkich zamówień.

- Po obmyśleniu planu, zdajesz sobie sprawę, że takie sprawdzenia powinno się wykonywać sekwencyjnie. Aplikacja może spróbować uwierzytelnić użytkownika otrzymawszy żądanie zawierające poświadczenia użytkownika. Jednak jeśli poświadczenia nie są prawidłowe i uwierzytelnienie nie powiedzie się, nie ma powodu dokonywać dalszych sprawdzeń.



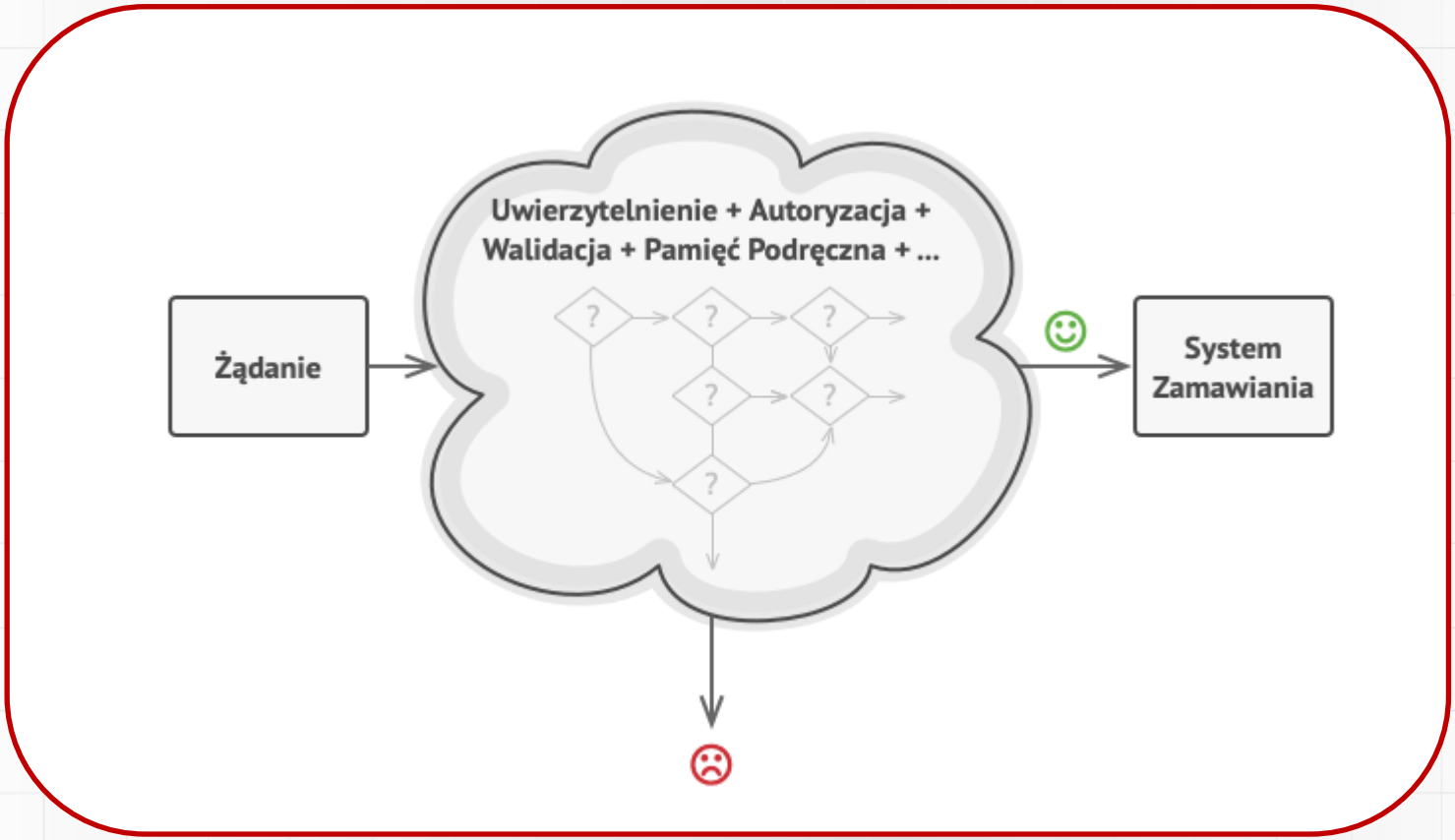
Wzorce behawioralne

W kolejnych miesiącach, implementujesz wiele takich sekwencyjnych sprawdzeń.

- Jeden z twoich współpracowników zauważa, że przekazywanie surowych danych przez system zamawiania nie jest bezpieczne. Dodajesz więc etap walidacyjny, czyszczący dane zawarte w żądaniu.
- Później ktoś zauważa, że system jest podatny na łamanie haseł metodą brute force. Aby się przed tym uchronić, dodajesz sprawdzenie odrzucające wielokrotne nieskuteczne próby uwierzytelnienia przychodzące z tego samego adresu IP.
- Ktoś inny zaś zasugerował, że można przyspieszyć działanie systemu, gdyby zwracał on przechowane w pamięci podręcznej wyniki żądań zawierające te same dane. Dodajesz więc kolejne sprawdzenie, pozwalające żądaniu przejść dalej tylko jeśli nie ma już stosownej odpowiedzi zapisanej w pamięci podręcznej.

Wzorce behawioralne

Im bardziej kod urósł, tym bardziej stał się zabałaganiony.



Wzorce behawioralne

- Kod sprawdzeń, który już na początku wyglądał pogmatwanie, spuchł jeszcze bardziej wraz z dodawaniem funkcjonalności. Zmiana jednego sprawdzenia czasem wpływała na inne. A co najgorsze, próba ponownego użycia sprawdzeń w zabezpieczeniu innych komponentów systemu spowodowała duplikację części kodu ponieważ niektóre komponenty potrzebowały sprawdzeń, ale inne nie.

- System stał się trudny do zrozumienia i kosztowny w utrzymaniu. Po okresie trudzenia się postanawiasz dokonać refaktoryzacji całości.

Wzorce behawioralne

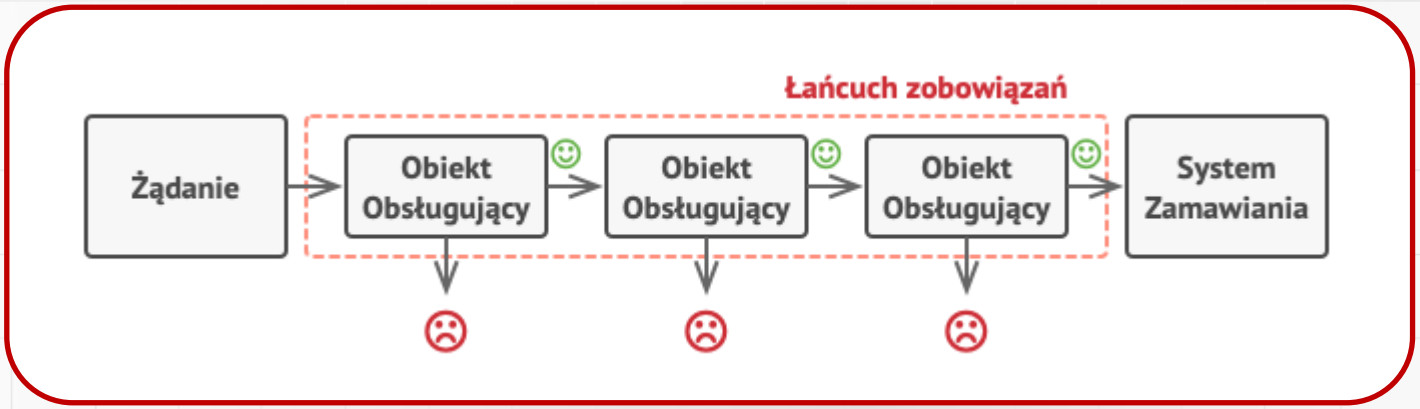
Rozwiązanie:

- Jak wiele innych wzorców behawioralnych, Łańcuch Zobowiązań zakłada przekształcenie pewnych obowiązków w samodzielne obiekty zwane obiektami obsługującymi. W naszym przypadku, każde sprawdzenie powinno się wyekstrahować do osobnej klasy posiadającej jedną metodę dokonującą sprawdzenia. Żądanie wraz z towarzyszącymi mu danymi przekazywane jest jako argument tej metody.
- Wzorzec sugeruje połączenie tych obiektów obsługujących w łańcuch. Każdy obiekt obsługujący stanowiący ogniwo łańcucha posiada pole przechowujące odniesienie do następnego obiektu w łańcuchu. Poza przetworzeniem żądania, obiekty przekazują je dalej. Żądanie biegnie wzdłuż łańcucha, by wszystkie ogniwa miały okazję je obsłużyć.A co najlepsze, obiekt obsługujący może zdecydować o nieprzekazaniu żądania dalej i tym samym kończy proces.
- W naszym przykładzie systemu zamawiającego, obiekt obsługujący dokonuje przetwarzania żądania i decyduje o przekazaniu go dalej, lub nie. Zakładając, że żądanie zawiera właściwe dane, obiekty obsługujące mogą wykonywać swoje obowiązki, takie jak uwierzytelnianie czy zapis w pamięci podręcznej.

Wzorce behawioralne

Rozwiązanie:

- W naszym przykładzie systemu zamawiającego, obiekt obsługujący dokonuje przetwarzania żądania i decyduje o przekazaniu go dalej, lub nie. Zakładając, że żądanie zawiera właściwe dane, obiekty obsługujące mogą wykonywać swoje obowiązki, takie jak uwierzytelnianie czy zapis w pamięci podręcznej.



Wzorce behawioralne

Rozwiązanie:

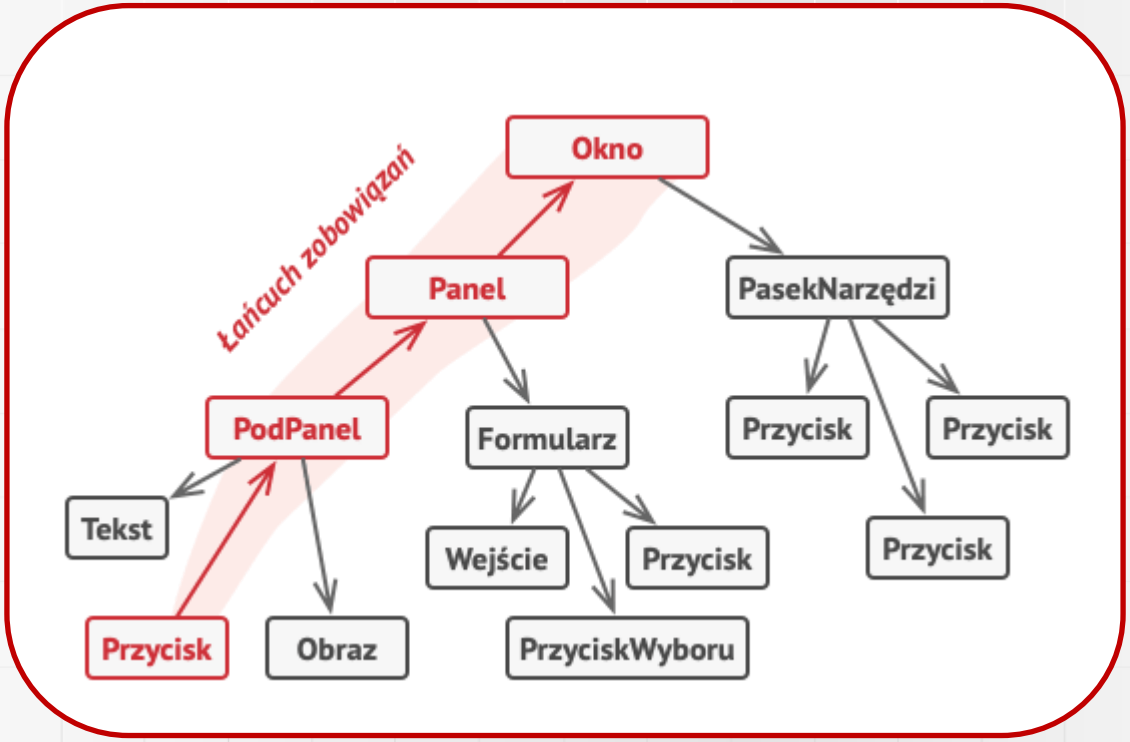
- Istnieje jednak nieco inne podejście (które weszło do kanonu), według którego obiekt obsługujący otrzymawszy żądanie decyduje czy może je obsłużyć i jeśli tak, to nie przekazuje go dalej. Więc albo tylko jeden obiekt obsługuje jedno żądanie, albo żaden. Podejście to jest bardzo powszechne w przypadku stosu zdarzeń w obrębie graficznego interfejsu użytkownika.

- Na przykład, gdy użytkownik kliknie przycisk, zdarzenie rozpropaguje się wzdłuż łańcucha elementów UI, zaczynając od przycisku, poprzez jego kontenery (formatki lub panele) i dociera do głównego okna aplikacji. Zdarzenie jest przetwarzane przez pierwszy element w łańcuchu który jest w stanie je obsłużyć. Ten przykład jest też godny uwagi, bo pokazuje jak z każdego drzewa obiektów można wyekstrahować łańcuch.

Wzorce behawioralne

Rozwiązanie:

- Istotnym jest, że wszystkie klasy obiektów obsługujących implementują ten sam interfejs. Każdy konkretny obiekt obsługujący powinien wiedzieć tylko o następnym, posiadającym metodę wykonaj. W ten sposób można komponować łańcuchy w trakcie działania programu, stosując różne obiekty obsługujące bez sprzęgania kodu z ich konkretnymi klasami.



Wzorce behawioralne

Analogia do prawdziwego życia:

- Właśnie kupiłeś sobie i zainstalowałeś jakąś część do komputera. Ponieważ jesteś geekiem, na komputerze jest kilka systemów operacyjnych. Uruchamiasz więc jeden po drugim, sprawdzając czy urządzenie jest obsługiwane. Windows wykrywa i włącza urządzenie automatycznie. Jednak twoja ukochana dystrybucja Linuksa odmawia współpracy. W nikłym przebłysku nadziei, dzwonisz na numer pomocy technicznej podany na opakowaniu.

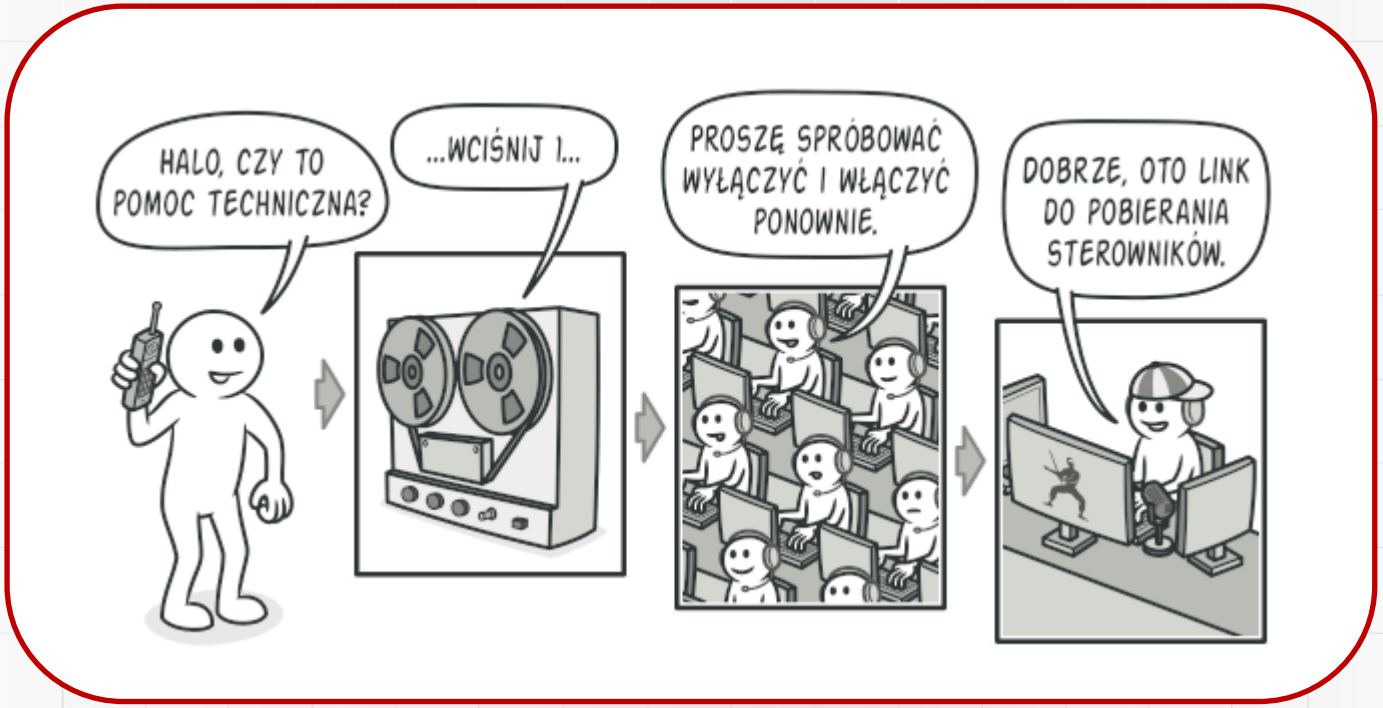
- Pierwsze, co słyszysz, to sztucznie brzmiący głos automatu zgłoszeniowego. Sugeruje on dziewięć typowych rozwiązań różnych problemów, ale żaden z nich nie dotyczy twego przypadku. Po jakimś czasie, automat poddaje się i łączy cię z żywym człowiekiem.

- Ale żywy pracownik również nie jest w stanie zasugerować nic pożytecznego. Cytuje długie ustępy instrukcji obsługi i nie słucha twoich uwag. Po usłyszeniu dziesiąty raz sugestii “proszę spróbować wyłączyć i włączyć ponownie komputer”, żądasz połączenia z prawdziwym inżynierem.

Wzorce behawioralne

Analogia do prawdziwego życia:

- Ostatecznie łączy cię z jednym z inżynierów, który zapewne od wielu godzin tęskni za rozmową z żywym człowiekiem, siedząc w swojej odosobnionej serwerowni gdzieś w ciemnej piwnicy. Inżynier podaje ci link do odpowiednich sterowników do urządzenia i tłumaczy jak je zainstalować pod Linuksem. Wreszcie — rozwiązanie! Rozłączasz się pełen radości.



Wzorce behawioralne

Zastosowanie:

- Stosuj wzorzec Łańcuch zobowiązań gdy twój program ma obsługiwać różne rodzaje żądań na różne sposoby, ale dokładne typy żądań i ich sekwencji nie są wcześniej znane.

- Stosuj ten wzorzec gdy istotne jest uruchomienie wielu obiektów obsługujących w pewnej kolejności.

Wzorce behawioralne

Zalety

- Można ustalać porządek obsługi żądania.
- Zasada pojedynczej odpowiedzialności. Można rozprzęgnąć klasy wywołujące działania klas od klas wykonujących działania.
- Zasada otwarte/zamknięte. Można wprowadzać do programu nowe obiekty obsługujące bez psucia istniejącego kodu klienta.

Wady

- Niektóre żądania mogą wcale nie zostać obsłużone.

Politechnika Wrocławska

Dziękuje bardzo za uwagę

Dr inż. Paweł Maślak