Krzysztof Regulski
AGH, WIMiIP

WZORCE

Wybór najważniejszych informacji

- **Refaktoryzacja (refactoring)** – Poprawa projektu struktury klas i samych klas bez zmiany funkcjonalności. Refaktoryzacja sprawia, że możliwe staje się ewolucyjne tworzenie oprogramowania, które pozostaje czytelne i proste. Refaktoryzacja ma często prowadzić do uzyskania struktury kodu zgodnej z wzorcami projektowania.
- "Każdy **wzorzec** opisuje problem, który ciągle pojawia się w naszej dziedzinie, a następnie określa zasadniczą część jego rozwiązania w taki sposób, by można było zastosować je nawet milion razy za każdym razem w nieco inny sposób"
- Zamiast skupiać się na funkcjonowaniu poszczególnych elementów, wzorce projektowe stanowią abstrakcyjny opis **zależności pomiędzy klasami**, co w efekcie wprowadza pewną standaryzację kodu oraz zwiększa jego zrozumiałość, wydajność i niezawodność.
- Wartość wzorców projektowych stanowi nie tylko samo rozwiązanie problemu, ale także dokumentacja, która wyjaśnia **cel, działanie, zalety** danego rozwiązania, co pomaga w łatwiejszym stosowaniu i adaptacji wzorców w danym zastosowaniu.
- Wzorzec składa się z trzech obszarów:
  - kontekst – warunki użycia wzorca
  - układ sił – zagadnienia związane z wzorce, cel, dziedzina problemu
  - rozwiązanie – konfiguracja klas, która rozwiązuje problem lub równoważy układ sił
- Wzorzec projektowy jest opisany przez:
  - nazwę - lakoniczny opis istoty wzorca
  - klasyfikację - kategorię, do której wzorzec należy
  - cel - do czego wzorzec służy
  - aliasy - inne nazwy, pod którymi jest znany
  - motywację - scenariusz opisujący problem i rozwiązanie
  - zastosowania - sytuacje, w których wzorzec jest stosowany
  - strukturę - graficzną reprezentację klas składowych wzorca
  - uczestników - nazwy i odpowiedzialności klas składowych wzorca
  - współdziałania - opis współpracy między uczestnikami
  - konsekwencje - efekty zastosowania wzorca
  - implementację - opis implementacji wzorca w danym języku
  - przykład - kod stosujący wzorzec
  - pokrewne wzorce - wzorce używane w podobnym kontekście
- Uproszczony opis wzorca zawiera:
  - nazwę wzorca
  - opis sytuacji, w której stosujemy wzorzec (motywacja do jego stosowania, problem, który wzorzec rozwiązuje) opis wzorca w postaci struktury klas i obiektów, ich wzajemnych zależności i interakcji (najczęściej w postaci diagramów UML)
  - opis uwarunkowań stosowania wzorca (wady i zalety, sposoby konkretnej implementacji, przykłady)

WZORCE KREACYJNE

(PSK - projektowanie systemów komputerowych, notatki w Internecie, Beata Frączek, http://brasil.cel.agh.edu.pl/~09sbfraczek)

wzorce kreacyjne - dotyczą sposobów tworzenia obiektów, skupiając się na abstrakcji i hermetyzacji tego procesu
Metoda Wytwórcza (Factory Method), Budowniczy (Builder), Fabryka Abstrakcyjna (Abstract Factory), Prototyp (Prototype), Singelton

1) Wzorzec Budowniczy (ang. Builder)

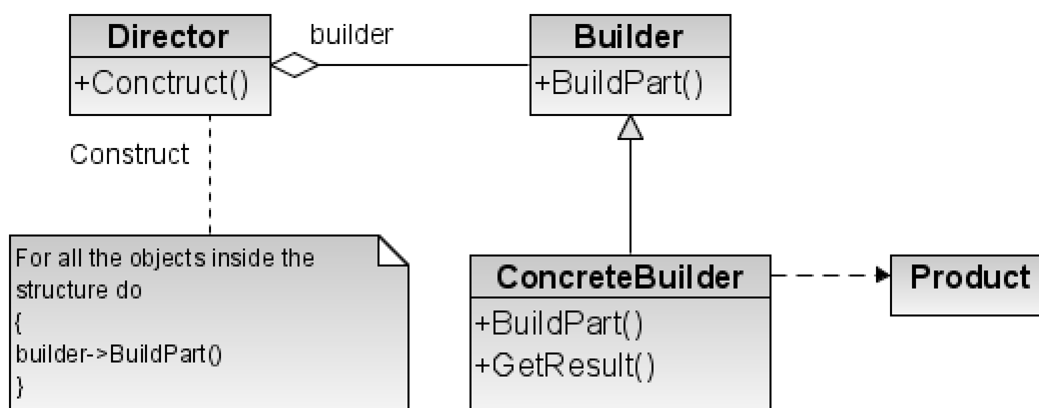
Przeznaczenie

Wzorzec ten oddziela konstrukcję obiektów złożonych od ich reprezentacji, umożliwiając tym samym powstawanie w jednym procesie konstrukcyjnym różnych reprezentacji.
Przykład: Czytnik plików RTF.

Przykładem może być tutaj oprogramowanie konwertujące tekst z jednego formatu na drugi. Algorytm odczytujący i interpretujący dane wejściowe jest oddzielony od algorytmu tworzącego dane wyjściowe. Dzięki takiemu rozwiązaniu możemy stosować jeden obiekt odczytujący dane wejściowe oraz wiele obiektów zapisujących je w różnych formatach (ASCII, HTML, RTF, itp.)

Budowniczy jest to jeden z kreacyjnych wzorców projektowych (obiektowy), którego celem jest rozdzielenie sposobu tworzenia obiektów od ich reprezentacji. Dzięki takiemu rozwiązaniu w tym samym procesie konstrukcyjnym możemy tworzyć różne reprezentacje obiektów. Budowniczy różni się od wzorca fabryki abstrakcyjnej oraz pozostałych wzorców kreacyjnych tym, że skupia się na sposobie tworzenia obiektów reprezentujących produkty. Budowniczy tworzy drobną cześć skomplikowanego produkt za każdym swoim wywołaniem jednocześnie kontrolując stan wykonanej pracy. Klient dostaje produkt po zakończeniu pracy Budowniczego a nie - tak jak w przypadku Fabryki abstrakcyjnej - "od razu".

Struktura

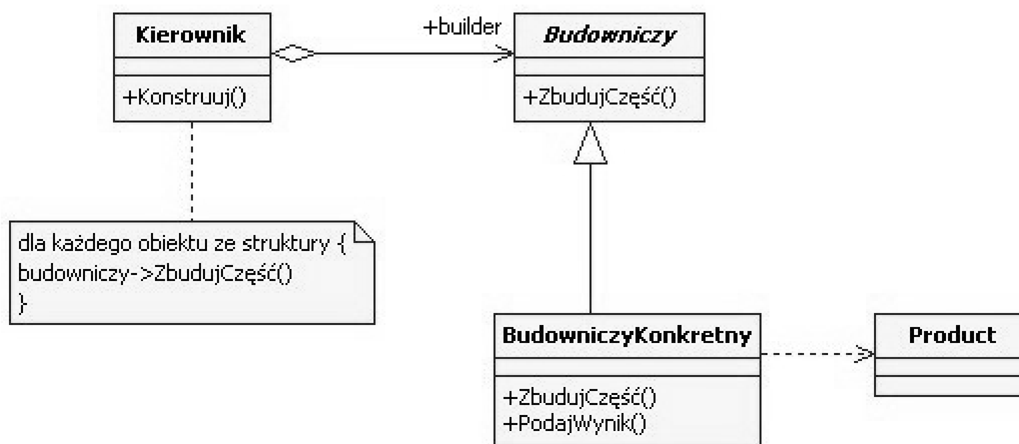


Jak widać na przedstawionym diagramie klas wzorzec składa się z dwóch podstawowych obiektów. Pierwsza z nich oznaczony jest jako Budowniczy - jego celem jest dostarczenie interfejsu do tworzenia produktów. Drugim obiektem jest obiekt oznaczony jako Konkretny Budowniczy a jego celem jest tworzenie konkretnych produktów korzystając z interfejsu obiektu Budowniczy. Strukturę wzorca uzupełnia obiekt Kierownika, wydaje on polecenia konstrukcji produktów wykorzystując do tego obiekt Budowniczego

Konsekwencje stosowania

Do plusów stosowania wzorca należą: duża możliwość zróżnicowania wewnętrznych struktur klas oraz możliwość kontrolowania tworzenia obiektów po stronie klienta. Minusem jest duża liczba obiektów reprezentujących konkretne produkty.

Uczestnicy

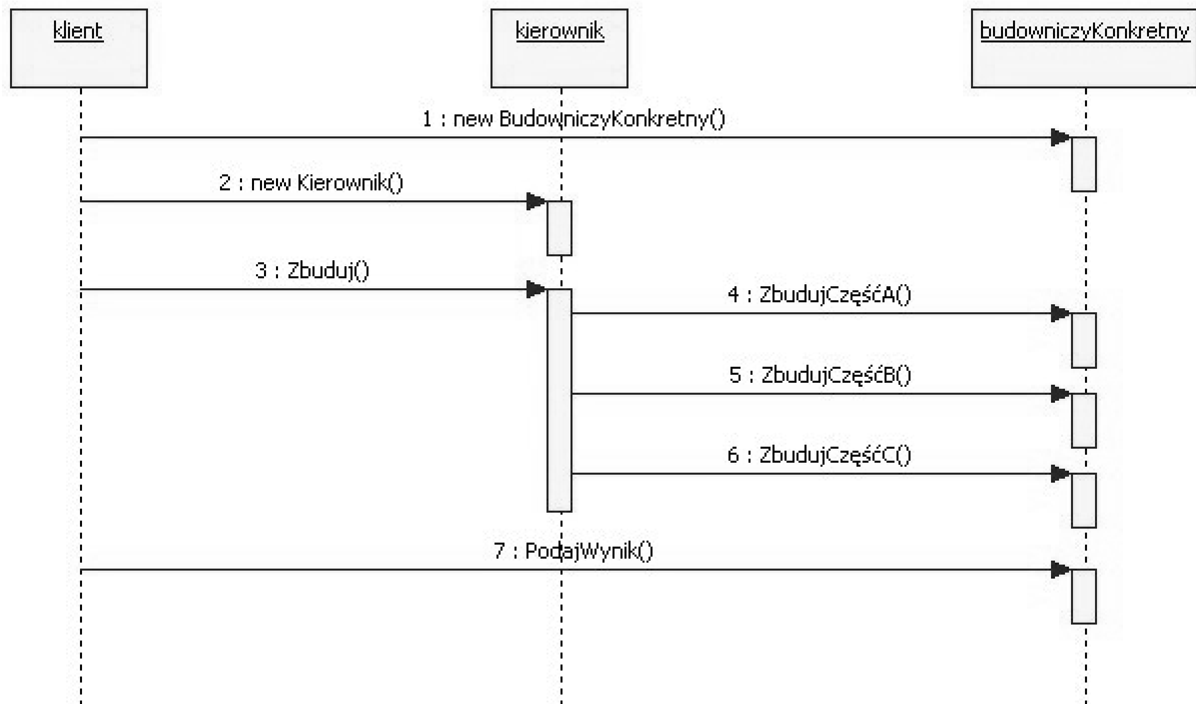


- Budowniczy - określa interfejs abstrakcyjny do tworzenia części składowych obiektu Produkt.
- BudowniczyKonkretny - konstruuje i zestawia części produktu poprzez implementowanie interfejsu Budowniczego; definiuje i kontroluje tworzoną przez siebie reprezentację; zapewnia interfejs do wyszukiwania produktu.
- Kierownik - konstruuje obiekt, używając interfejsu Budowniczego.
- Produkt - reprezentuje konstruowany obiekt złożony; Klasa BudowniczyKonkretny buduje wewnętrzną reprezentację produktu i definiuje proces składania go; zawiera klasy definiujące części składowe, włączając w to interfejsy zestawiania części w końcowy wynik.

Współpraca

- Klient tworzy obiekt Kierownik i konfiguruje go za pomocą pożądanego obiektu Budowniczy.
- Kierownik informuje budowniczego o potrzebie zbudowania części produktu.
- Budowniczy przetwarza żądania kierownika i dodaje części do produktu.
- Klient odbiera produkt od budowniczego.

Diagram sekwencji



2) Fabryka abstrakcyjna, Fabryka abstrakcji, Abstract factory

Uzasadnienie stosowania:

Jedna metoda fabrykująca tworzy obiekty implementujące jeden interfejs. Dodając do polimorficznej wersji kolejne metody fabrykujące otrzymujemy fabrykę abstrakcyjną, zdolną do tworzenia obiektów implementujących różne interfejsy. Tworząc konkretną fabrykę decydujemy tylko raz o typach wszystkich konkretnych implementacji (muszą więc one być ze sobą sensownie powiązane). Nie musimy podejmować osobnych decyzji dla każdego interfejsu (było by to konieczne przy użyciu kilku osobnych Metod fabrykujących zamiast jednej fabryki).

Intencja

Dostarcza możliwość tworzenia rodzin zależnych lub spokrewnionych obiektów bez opisywania konkretnych klas.

Motywacja

Załóżmy, że dostarczamy narzędzia do budowy interfejsu użytkownika, w którym użytkownik może zmieniać jego wygląd pomiędzy kilkoma standardami, np. Windows i RedHat Linux

Programowanie aplikacji powinno być niezależne od wyglądu, aby mogło być przenaszalne i łatwo rozszerzalne.

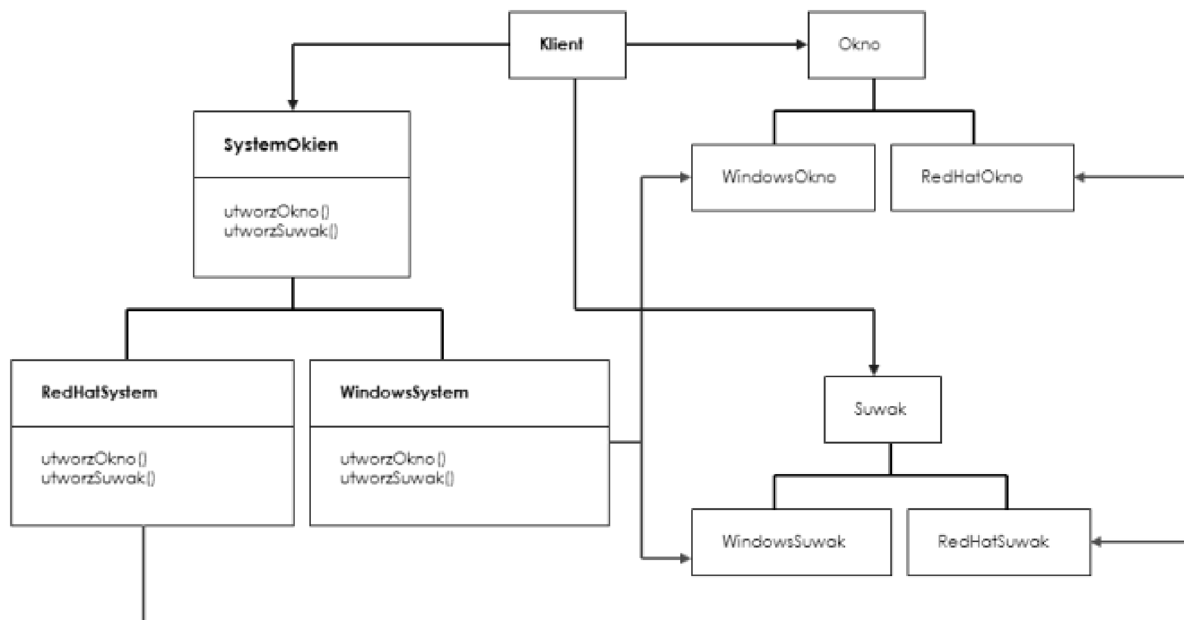
Stosowalność:

Gdy chcemy się uniezależnić od złożonego API. Obudowując zewnętrzne API systemu wrapperami (wzorzec Most) możemy stworzyć rodzinę implementacji różnych interfejsów która będzie odpowiadała za komunikację z danym nam API. Dodając obsługę nowego API do systemu dodajemy po jednej implementacji do każdego interfejsu i jedną konkretną fabrykę. W jednym tylko miejscu kodu decydujemy jakiego interfejsu chcemy użyć poprzez stworzenie jednej z konkretnych fabryk.

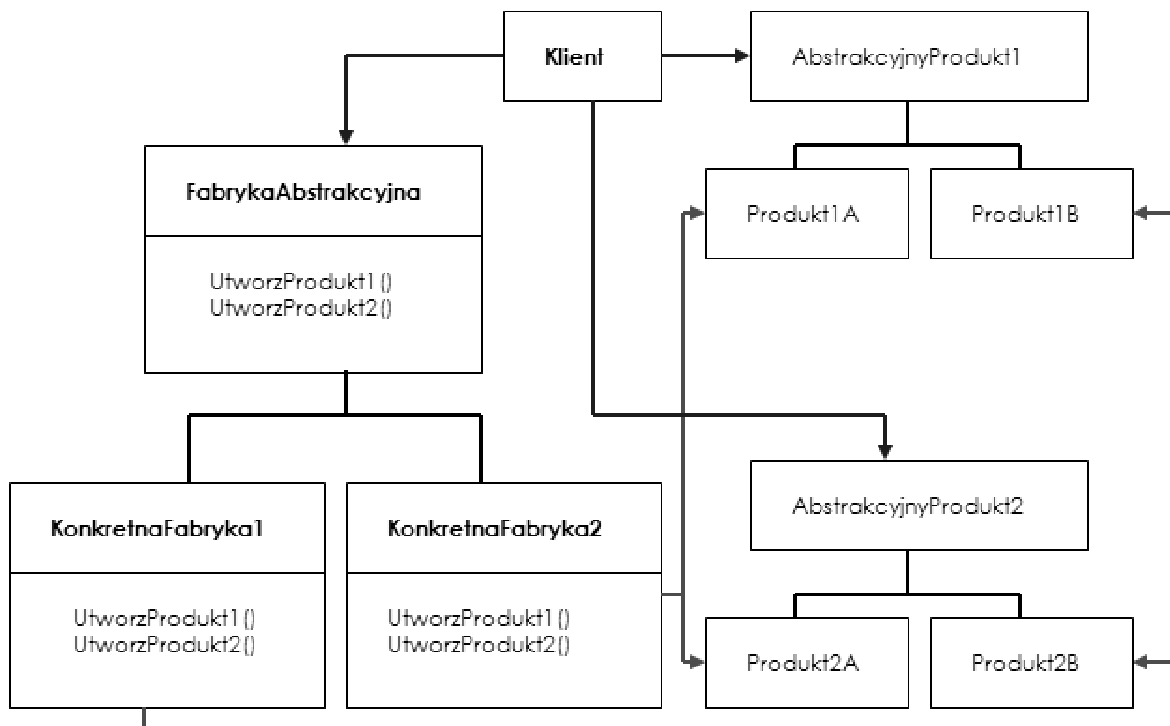
Użycie:

Tworze nowa metodę a w niej warunki - jeżeli parametr = windows to generuje nowy obiekt new WidowsSystem i tak dalej (jak w Fabryce). Rozwiązujemy nasz problem przez zdefiniowanie abstrakcyjnej klasy SystemOkien, która dostarcza interfejs do kreowania rożnych rodzajów narzędzi (tutaj okno, suwak). Dla każdego rodzaju narzędzia (okna, suwaka) istnieje również klasa abstrakcyjna, której konkretne podklasy implementują narzędzia dla określonego już wyglądu. SystemOkien posiada metody zwracające obiekt dla każdej abstrakcyjnej klasy narzędzi. Klient wywołuje te metody nie wiedząc, z której konkretnie klasy korzysta.

Przykładowy diagram



Struktura Fabryki Abstrakcyjnej



Elementy Fabryki Abstrakcyjnej

- **FabrykaAbstrakcyjna** (SystemOkien) Deklaruje interfejs dla operacji, które tworzą obiekty AbstrakcyjnyProdukt
- **KonkretnaFabryka** (WindowsSystem, RedHatSystem) Implementuje operacje do tworzenia obiektów KonkretnyProdukt
- **AbstrakcyjnyProdukt** (okno, suwak) Deklaruje interfejs dla typu produktu
- **KonkretnyProdukt** (WindowsOkno, WindowsSuwak, itd.) Implementuje interfejs AbstrakcyjnyProdukt
- **Klient**
  Używa tylko interfejsów FabrykaAbstrakcyjna i AbstrakcyjnyProdukt

Konsekwencje

Izoluje konkretne klasy. Ułatwia wymianę rodzin produktów

3) Metoda wytwórcza

Przeznaczenie

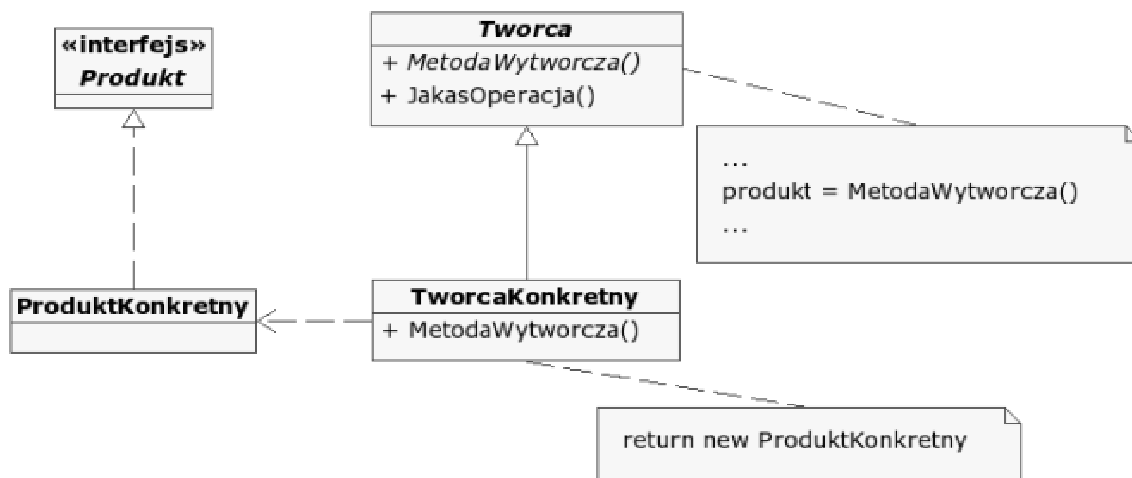
Wzorzec ten określa interfejs do tworzenia obiektów, lecz umożliwia podklasom decydowanie o tym, której klasy ma to być obiekt. Dzięki Metodzie Wytwórczej klasy mogą zdać się na podklasy w kwestii tworzenia egzemplarzy.

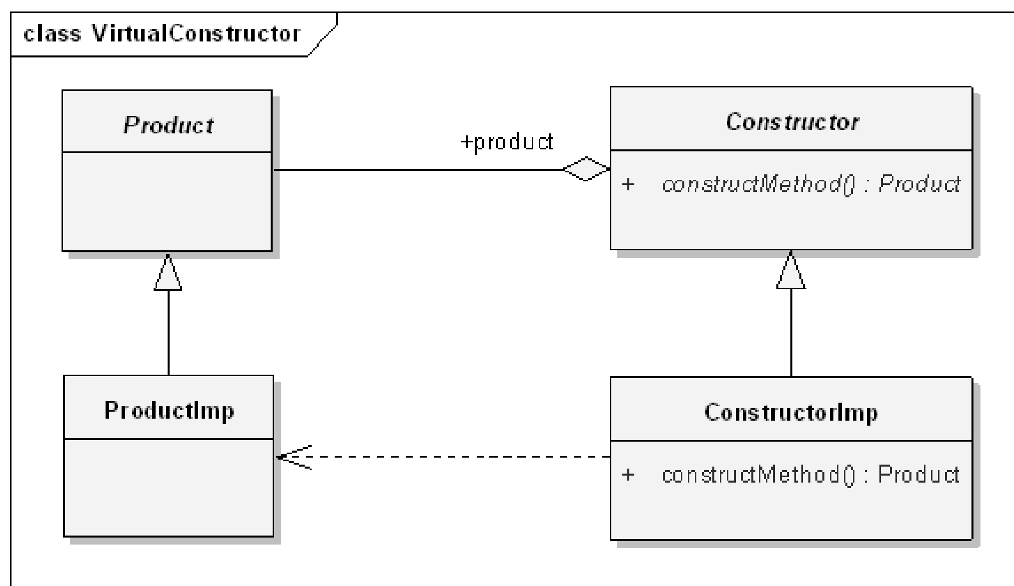
Przykład: Zrąb aplikacji MDI.

Stosowalność metod wytwórczych

- klasa nie może przewidzieć, jakich klas obiekty musi tworzyć
- klasa chce, by to jej podklasy specyfikowały tworzone przez nią obiekty
- klasy przekazuj? odpowiedzialność jednej z kilku podklas pomocniczych, a Ty chcesz tylko w jednym miejscu ustalić, która z podklas pomocniczych jest ich delegatem

Struktura metody wytwórczej





Metoda wytwórcza - implementacja

Dwa główne typy wzorca:

- z abstrakcyjnym twórcą - wymaga implementacji podklas dla konkretnych obiektów
- z domyślną implementacją metody tworzącej

Pozwala na łączenie kilku hierarchii klas (np. produkty i obiekty manipulujące)

Typy klas można parametryzować poprzez argumenty wejściowe metod wytwórczych

Implementacja metod wytwórczych zależy od wykorzystywanego języka programowania

Zastosowanie

- Metoda wytwórcza jest najczęściej stosowanym wzorcem w szkieletach aplikacji
- Najczęściej metodę wytwórczą można wykorzystać w implementacji fabryki abstrakcyjnej
- Większość sterowników, interfejsów, pakietów komponentów realizuje scenariusz inicjalizacji jak metodę wytwórczą

4) Prototyp

Intencja

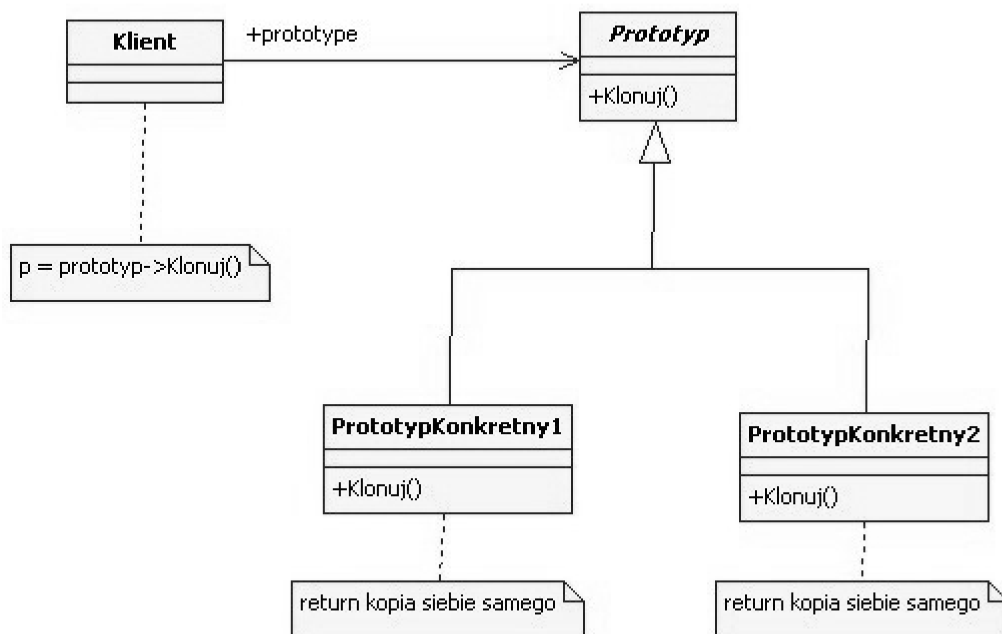
Kopiowanie instancji już istniejących i modyfikowanie ich, zamiast tworzenia całkiem nowych instancji

Stosowalność

Wzorzec Prototyp powinien być używany, gdy:

- system powinien być niezależny od tego, jak jego produkty są tworzone, składane i reprezentowane;
- klasy, których egzemplarze należy tworzyć są specyfikowane w czasie wykonywania programu, np. przez dynamiczne ładowanie
- istnieje potrzeba uniknięcia budowania hierarchii klas fabryk, która jest porównywalna z hierarchią klas produktów
- stan obiektów klasy może przyjmować tylko jedną z kilku różnych wartości; może być wówczas wygodniej zainstalować odpowiednią liczbę prototypów i klonować je niż ręcznie tworzyć egzemplarze klasy za każdym razem z odpowiednim stanem.

Struktura



Uczestnicy

- Prototyp - deklaruje interfejs klonowania się.
- PrototypKonkretny - implementuje operację klonowania się.
- Klient - tworzy nowy obiekt, prosząc prototyp o sklonowanie się.

Współpraca - Klient prosi prototyp o sklonowanie się.

Konsekwencje

Pozwala na dodawanie i usuwanie obiektów w czasie przebiegu programu Przyspiesza kreowanie dużych obiektów

5) Wzorzec projektowy Singleton

Przeznaczenie

- przeznaczony do ograniczania możliwości tworzenia obiektów danej klasy do jednej instancji oraz zapewnienie globalnego punktu dostępu do niej
- używany w sytuacji, gdy istnieje potrzeba stworzenia klasy, która posiadałaby wyłącznie jedną instancję
- użycie zamiast obiektu globalnego - nie zaśmiecany wtedy globalnej przestrzeni nazw różnymi nazwami obiektów

Motywacja

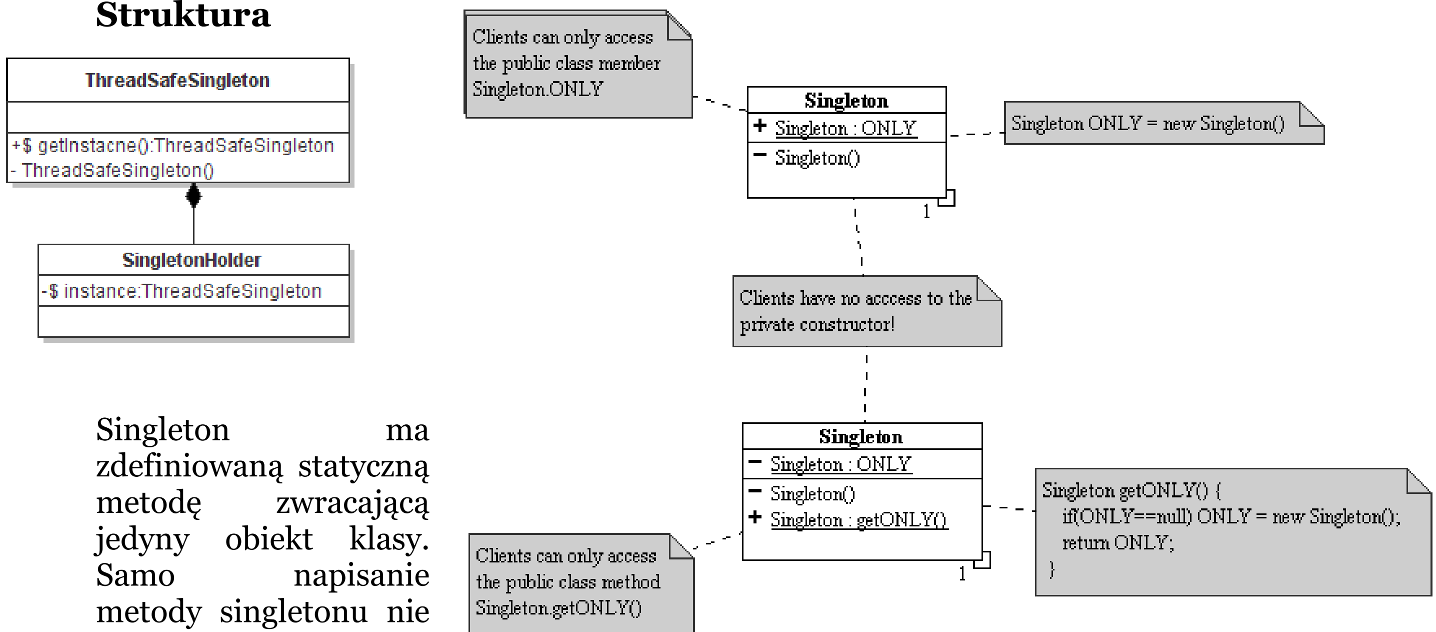
Singleton polega na stworzeniu klasy, ktora przechwytuje żądania powstania liczniejszych instancji,oraz zapewnia łatwy dostęp do tej jedynej istniejącej Jedyna instancja jest rozszerzalna dla jej podklas, w ktorych można jej używać bez modyfikacji kodu.

Przykłady użycia: w systemie może występować tylko jeden manager okien, jedyny punkt dostępu do baz danych, jedna kolejka do jednej drukarki.

Konsekwencje

- Kontrolowany dostęp do jedynej instancji
- Zredukowana przestrzeń nazw
- Pozwala na zwiększenie dozwolonej ilości instancji

Struktura



Singleton ma zdefiniowaną statyczną metodę zwracającą jedyny obiekt klasy. Samo napisanie metody singletonu nie wystarczy. Po napisaniu tej metody zmienia się sposób tworzenia instancji klasy. Zamiast: Singleton= new Singleton( ); Instancje tworzy się w ten sposób: Singleton = Singleton::getInstance ();

Zalety

Kontrolowany dostęp w odróżnieniu od obiektu globalnego. Gwarancja, że obiekt powstanie i zostanie zniszczony tylko raz - ta własność jest szczególnie ważna na systemów gdzie Singleton zarządza globalnym zasobem, np. połączeniem z bazą danych

Wady

Często nieumiejętnie stosowany: nie zrozumiałym jest tworzenie Singletonu zawierającego zmienne publiczne, lub stałe. Uzasadnione jest użycie Singletonu zawierającego stałe, gdy te stałe są wprowadzane w trakcie wykonywania programu. Gdy znamy wartości stałych przed wykonaniem należy użyć statycznych pól finalnych.

Inną powszechną wadą jest tworzenie zbyt dużych klas singletonów (tzw. Klas boskich) reprezentujących cały system, np. klasa system agregująca wszystkie klasy.

Przykład:

Klasycznym przykładem singletonu jest klasa Toolbox, posiadająca metodę getDefaultToolbox(), która zwraca egzemplarz klasy Toolbox