

# Paradygmaty Programowania

Marcin Bączyk

Wykład 7

12 kwietnia 2021

- wzorce projektowe
  - budowniczy
  - fabryka abstrakcyjna
  - kompozyt
  - dekorator
  - polecenie

- Robert C. Martin: “Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów”
- Robert C. Martin: “Zwinnej wytwarzanie oprogramowania. Najlepsze zasady, wzorce i praktyki”
- Grady Booch, James Rumbaugh, Ivar Jacobson: “UML przewodnik użytkownika”
- Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides: “Wzorce projektowe”

## Wprowadzenie

W programowaniu (zwłaszcza) obiektowym istnieją klasy problemów, z którymi programiści (projektanci) spotykają się stosunkowo często. Ze względu na ich powtarzalność opracowano szereg standardowych mechanizmów pozwalających na poprawne oraz stosunkowo szybkie rozwiązanie danego problemu.

## Wzorzec projektowy

*„Wzorzec projektowy jest sprawdzonym w praktyce rozwiązaniem często pojawiających się problemów i powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie modyfikację oraz utrzymanie kodu źródłowego. Jest opisem rozwiązania.”*

## Elementy wzorca projektowego

- nazwa - jednoznacznie identyfikuje konkretne rozwiązanie; w kodzie często pojawiają się nazwy klas odnoszące się do nazw wykorzystanych w dokumentacji wzorca;
- opis problemu - określa kiedy stosować dany wzorzec; wyjaśnia o co dokładnie chodzi w danym wzorcu;
- opis rozwiązania - określa elementy składające się na projekt, ich związki, zobowiązania i współpracę;
- konsekwencje stosowania - wyniki oraz wady i zalety stosowania danego wzorca.

## Podział wzorców projektowych ze względu na rodzaj

- kreacyjne - związane z procesem tworzenia obiektów
- strukturalne - dotyczą składania klas lub obiektów
- czynnościowe - charakteryzują sposób w jaki klasy lub obiekty oddziałują na siebie i rozdzielają zobowiązania (odpowiedzialność).

## Podział wzorców projektowych ze względu na zakres

- klasowe - zajmują się związkami między klasami i ich podklasami; związki ustalane statycznie;
- obiekty - zajmują się związkami między obiektami; związki ustalane dynamicznie

		Rodzaj		
		kreacyjne	strukturalne	czynnościowe
zakres	klasy	metoda wytwórcza	adapter	interpreter metoda szablonowa
	obiekty	<b>budowniczy</b> <b>fabryka abstrakcyjna</b> prototyp singleton	adapter <b>dekorator</b> fasada <b>kompozyt</b> most pełnomocnik pyłek	iterator łańcuch zobowiązań mediator <i>obserwator</i> odwiedzający pamiętka <b>polecenie</b> stan <b>strategia</b>

Wzorce projektowe:

relacji

- stanowią abstrakcyjny opis pomiędzy klasami;
- przyczyniają się do standaryzacji kodu;
- pozwalają lepiej komunikować programistom ich zamiary
- zwiększają czytelności i niezawodność kodu
- przyspieszają proces rozwoju oprogramowania;



## Przykładowy problem

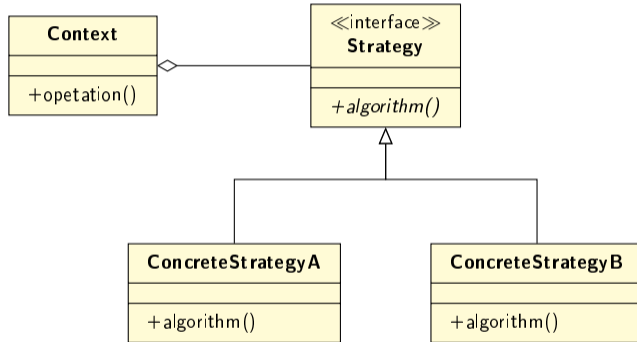
Dany jest symulator układów analogowych wraz ze wszystkimi klasami niezbędnymi do reprezentacji poszczególnych elementów dyskretnych.

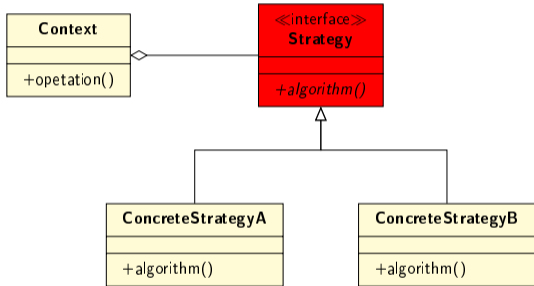
Cechy symulatora:

- Dla danego układu / podukładu istnieje możliwość wyznaczenia impedancji zastępczej.
- Istnieje możliwość wyboru metody jaką impedancja zastępcza jest wyznaczana.

Należy zaprojektować mechanizm, który pozwoli na przełączenie metody wyznaczania impedancji zastępczej danego układu bez wpływu na swoich klientów.

# wzorzec projektowy «strategia»

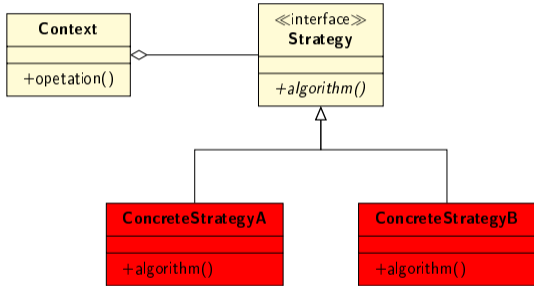




## Strategia

- Deklaruje interfejs wspólny dla wszystkich obsługiwanych algorytmów.

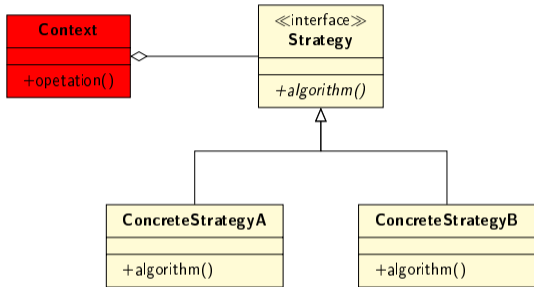
**Strategia** definiuje jaka metoda musi być zdefiniowana w poszczególnych **Strategiach Konkretnych**. **Kontekst** wykorzystuje ten interfejs do wykonywania algorytmu zdefiniowanego przez **Konkretną Strategię**.



## Strategia Konkretna

- Implementuje algorytm, wykorzystując interfejs z klasy **Strategia**.

**Kontekst** zawsze wykorzystuje obiekt klasy **Konkretna Strategia**.



## Kontekst

- Jest konfigurowany za pomocą obiektu klasy **Strategia Konkretna**.
- Utrzymuje odwołanie do obiektu typu **Strategia**.
- Może zdefiniować interfejs umożliwiający **Strategii** uzyskanie dostępu do jego danych.

W niektórych przypadkach **Kontekst** może zarządzać / być konfigurowanym wieloma obiektami różnych klas **Konkretna Strategia**.

Obiekty klas **Strategia** i **Kontekst** współdziałają w celu zaimplementowania wybranego algorytmu. Dane niezbędne do wykonania algorytmu mogą być przekazane w trakcie wywołania lub później poprzez referencję do obiektu **Kontekstu**.

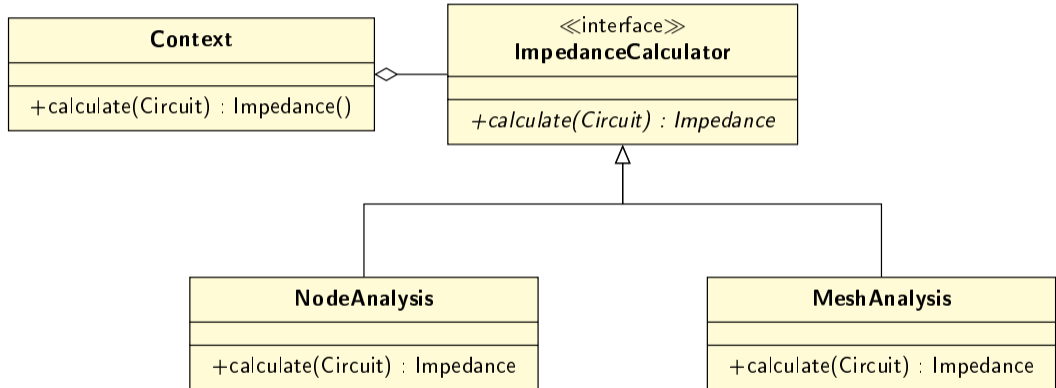
**Kontekst** przekazuje żądania od swoich klientów do konkretnej(-ch) strategii. Klienci mogą bezpośrednio konfigurować obiekt **Kontekstu** lub posługiwać się obiektem skonfigurowanym przez innych aktorów.

## Obszar zastosowań

Stosuj wzorzec strategii, gdy:

- wiele powiązanych ze sobą klas różni się jedynie zachowaniem;
- potrzebne są różne warianty jakiegoś algorytmu;
- w algorytmie są używane dane, o których klienci nie powinni wiedzieć;
- klasa definiuje wiele zachowań, które w operacjach są uwzględnione w postaci wielokrotnych instrukcji warunkowych.

# wzorec projektowy «strategia»





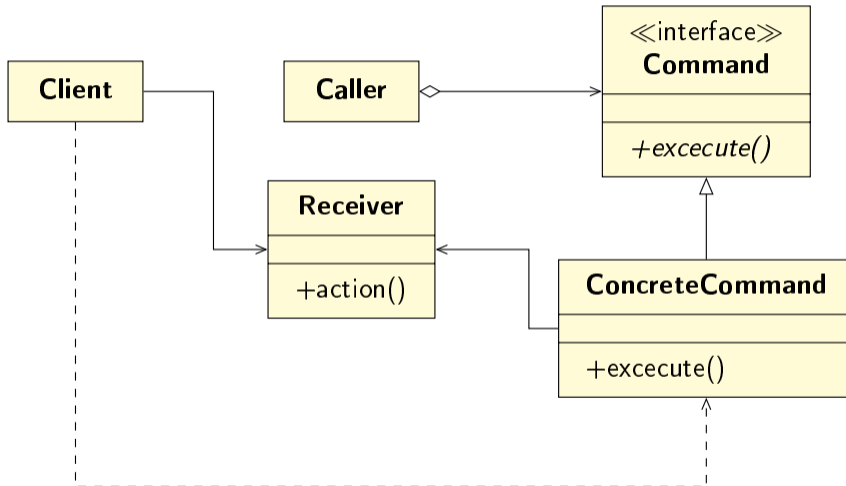
### Przykładowy problem

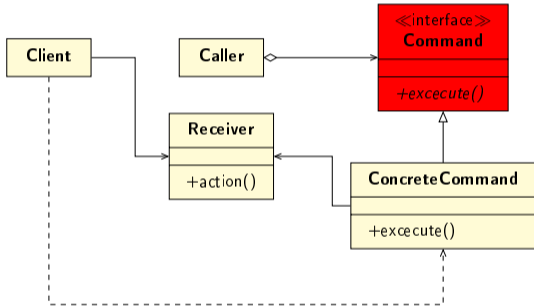
Dany jest graficzny edytor dla symulatora układów elektronicznych złożonych z elementów dyskretnych.

Cechy edytora:

- Pozwala na zapamiętanie ostatnio wykonanych akcji i umożliwia ich cofanie.
- Pozwala również na ponawianie wcześniej cofniętych akcji.

Należy zaprojektować mechanizm pozwalający w prosty sposób zaimplementować cofanie oraz ponawianie wprowadzanych zmian.

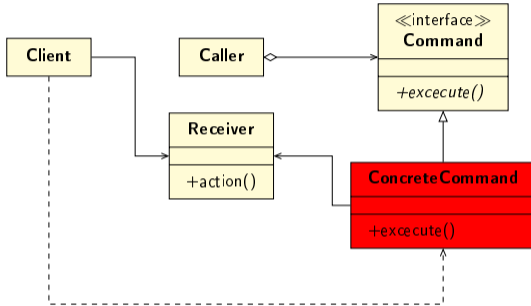




## Polecenie

- Definiuje interfejs do wykonywania operacji.

**Polecenie** może deklarować jedynie metodę do jednorazowego wykonania działania. Jeżeli to możliwe i uzasadnione **Polecenie** może również zawierać deklarację metody do cofania zmian oraz ich ponownego wykonywania.

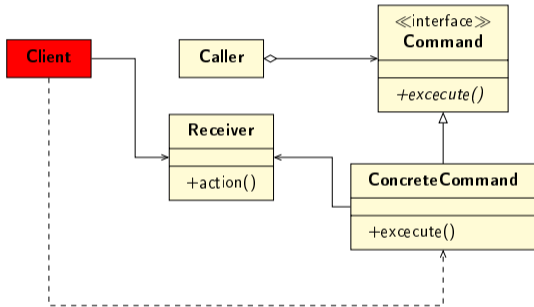


## KonkretnyPolecenie

- Definiuje powiązanie między **Odbiorcą** i akcją.
- Implementuje operację wykonaj wykorzystując odpowiednie operacje **Odbiorcy**.

**KonkretnyPolecenie** implementuje konkretne działanie, które może być odłożone w czasie (nawet jeżeli ten czas jest bardzo krótki). Zazwyczaj, jeżeli w systemie implementowany jest ten wzorec to istnieje bardzo dużo różnych implementacji **KonkretnychPoleceń**.

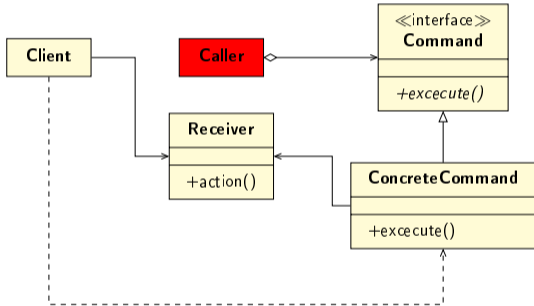
W ramach **KonkretnychPoleceń** często implementuje się również wzorec kompozyt w celu rejestracji bardziej złożonych poleceń.



## Klient

- Tworzy **KonkretnoPolecenie** i ustala jego **Odbiorcę**.

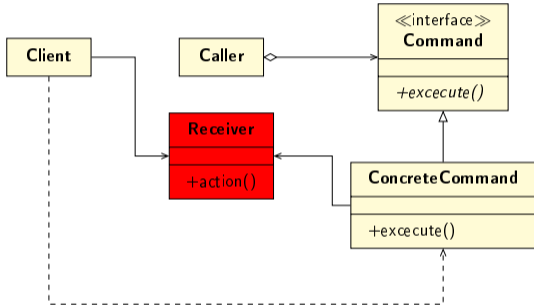
**Klient** ustala, na którym obiekcie będzie wykonywane polecenie przekazując referencję do **Odbiorcy KonkretnemuPoleceniu**. Utworzone **KonkretnoPolecenie** **Klient** przekazuje **Wywołującemu**.



## Wywołujący

- Prosi **Polecenie** o spełnienie żądania.

**Wywołujący**, jeżeli jest implementowany mechanizm cofania poleceń, zazwyczaj przechowuje **Polecenia** w kolekcji umożliwiającej przechodzenie sekwencyjne po jej elementach.



## Odbiorca

- Wie jak wykonać konkretne operacje związane ze spełnieniem żądania.

**Odbiorca** najczęściej jest obiektem, który ulega zmianie w związku z wykonywanym **KonkretnymPoleceniem**. **Odbiorca** implementuje całą funkcjonalność związaną ze spełnieniem żądania oraz (jeśli to możliwe i uzasadnione) jego cofnięciem.

Klient tworzy **KonkretnePolecenie** przekazując mu **Odbiorcę** tego polecenia a następnie przekazuje je **Wywołującemu**. **Wywołujący** przechowuje **Polecenie** tak długo jak jest mu ono potrzebne. **Wywołujący** zgłasza żądanie, wywołując operację `wykonaj()` **Polecenia**. **KonkretnePolecenie** wywołuje operacje swojego **Odbiorcy** w celu spełnienia żądania.



## Obszar zastosowań

Stosuj wzorzec polecenia, gdy:

- w systemie istnieje potrzeba, aby niektóre akcje mogły być enkapsulowane w postaci obiektów w celu ich późniejszego wykonania;
- okres życia tych obiektów zazwyczaj jest niezależny od pierwotnego żądania wykonania akcji;
- specyfikowanie żądań i ich wykonanie może być rozdzielone w czasie;
- istnieje potrzeba logowania wykonywanych akcji, kolejkowania;
- istnieje potrzeba wprowadzenia możliwości anulowania poleceń.

## Przykładowy problem

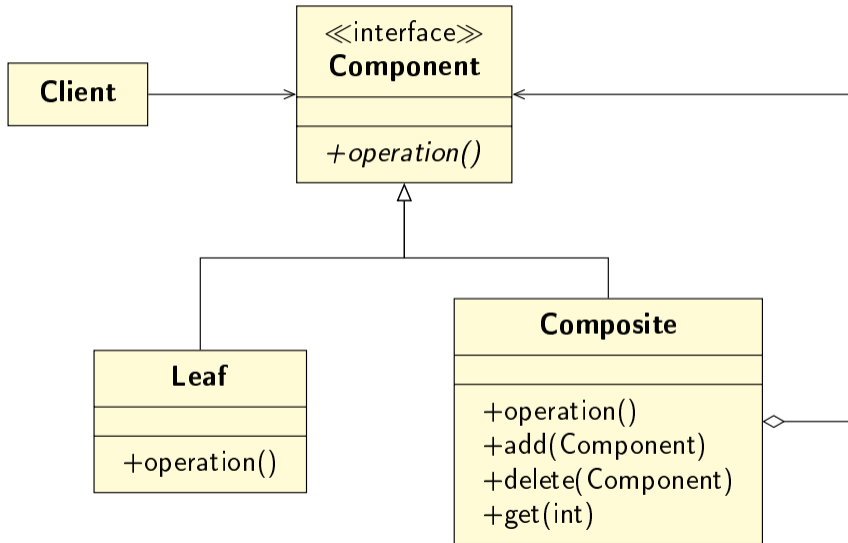
Dany jest symulator układów analogowych wraz ze wszystkimi klasami niezbędnymi do reprezentacji poszczególnych elementów dyskretnych.

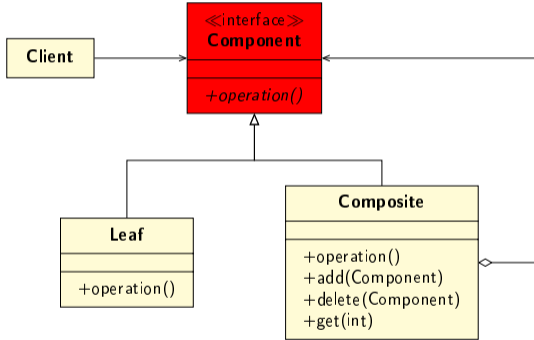
Cechy symulatora:

- Pozwala na budowanie podukładów będących dwójnikami, które później mogą być wielokrotnie wykorzystywane do budowania bardziej złożonych systemów.
- Budowane układy i podukłady mogą być wielokrotnie zagnieżdżane.

Należy zaprojektować taką strukturę reprezentacji poszczególnych elementów i układów w systemie, aby symulator nie musiał ingerować w to z jakiego rodzaju elementem czy podukładem ma do czynienia.

# wzorec projektowy «kompozyt»

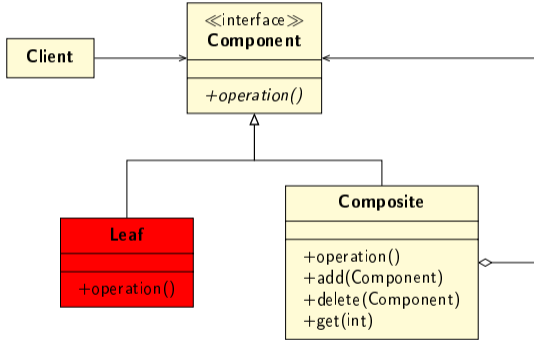




## Komponent

- Deklaruje interfejs składowych obiektów.

Klienci używając poszczególnych komponentów wykorzystują metody zdefiniowane w klasie bazowej.

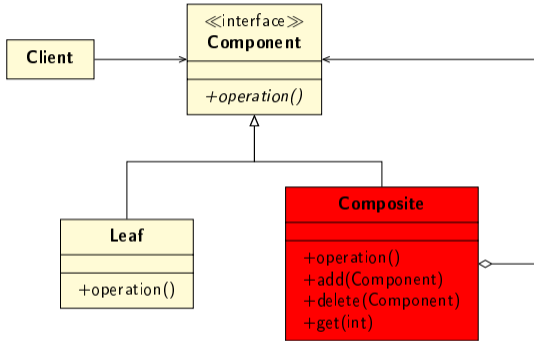


## Liść

- Definiuje działanie obiektów pierwotnych w strukturze.

W danej strukturze może istnieć wiele różnych klas będących liśćmi. Poszczególne liście mogą implementować zachowanie właściwe dla swojego rodzaju.

# wzorec projektowy «kompozyt»



## Kompozyt

- Definiuje działanie **Komponentów** składających się z innych **Komponentów**.
- Przechowuje **Komponenty** z których jest złożony.
- Udostępnia operacje związane z zarządzaniem składowymi.

Gdy wywołanie konkretnej operacji zdefiniowanej w Komponentie zwraca wynik, Kompozyt składa wyniki działania swoich Liści w spójną całość. Dotyczy to również tak zwanych efektów ubocznych.

Poszczególni **Klienci** nie wiedzą czy mają do czynienia z pojedynczym obiektem czy całą strukturą. W ten sposób uproszczona zostaje ich budowa.

Dodawanie nowych rodzajów **Komponentów** jest proste. W językach o statycznym typowaniu (C++) dodanie nowego typu **Komponentu** nie wymaga ponownej kompilacji kodu **Klientów**.

Istnieją implementacje wzorca projektowego kompozyt, którym klasa **Komponentu** implementuje również operacje związane z zarządzaniem składowymi klasy **Kompozytu**: W takim przypadku **Klienci** również mają możliwość modyfikowania drzewiastej struktury **Kompozytu**. Od projektanta systemu zależy, która implementacja będzie wykorzystywana.

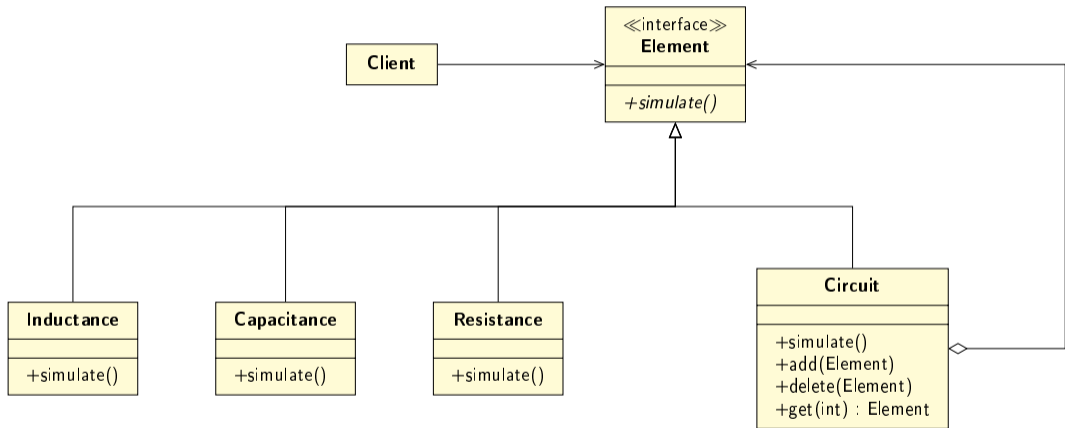
## Obszar zastosowań

Stosuj wzorzec kompozytu, gdy

- w systemie istnieją rodziny obiektów, które mogą być przedstawiane w postaci hierarchii;
- klienci wykorzystujący obiekty powinni móc ignorować różnicę między prostym obiektem a złożonym.



# wzorec projektowy «kompozyt»



## Przykładowy problem

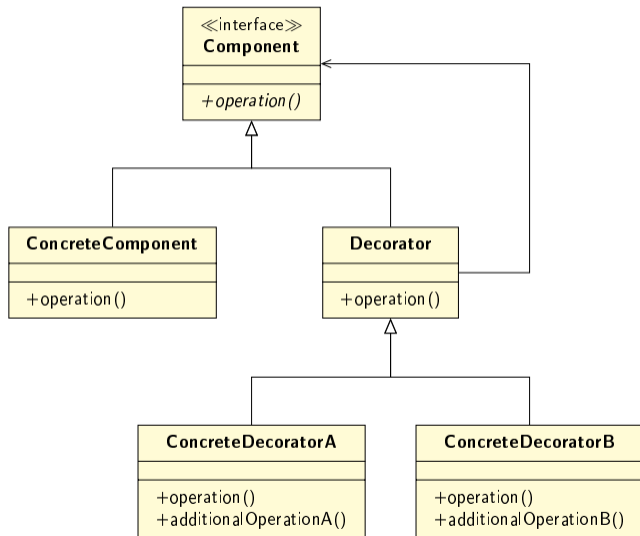
Dany jest symulator układów analogowych wraz ze wszystkimi klasami niezbędnymi do reprezentacji poszczególnych elementów dyskretnych.

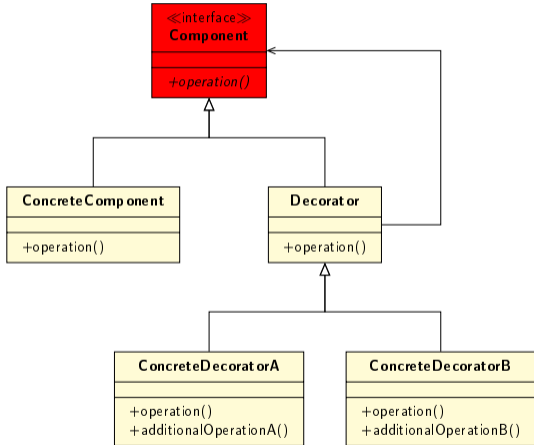
Cechy symulatora:

- Dla symulacji czasowej, pozwala wybrać elementy dla, których rejestrowane jest napięcie na zaciskach oraz prąd przepływający przez te element.
- Mechanizmy rejestracji mogą być dodawane niezależnie od siebie oraz niezależnie od wybranego typu elementu.

Należy zaprojektować mechanizm, który niezależnie od rodzaju elementu lub podukładu pozwoli na dynamiczne dodawanie do obiektu opcji rejestracji napięcia na zaciskach tego elementu lub prąd przez niego przepływający.

# wzorzec projektowy «dekorator»

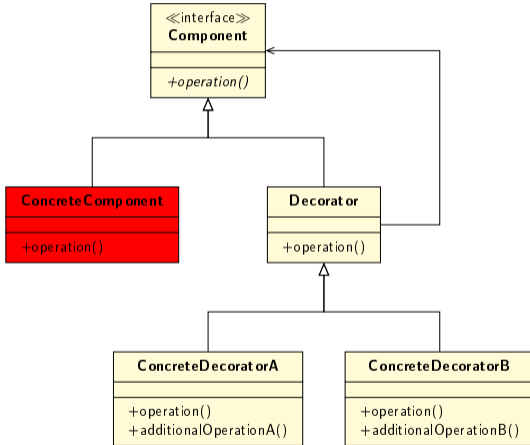




## Komponent

- Deklaruje interfejs obiektów, do których można dynamicznie dołączyć zobowiązanie.

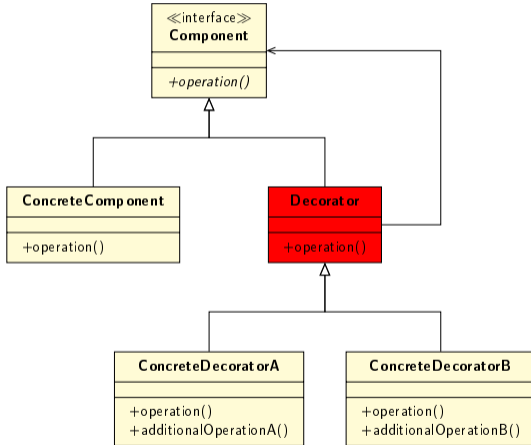
**Komponent** definiuje jakie metody muszą być zdefiniowane w **Konkretnych Komponentach** oraz poszczególnych **Dekoratorach**. Klienci korzystający z **Komponentów** nie mają świadomości czy korzystają **Konkretny Komponent** został udekorowany czy nie.



## KonkretnyKomponent

- Definiuje obiekt, do którego można dołączyć dodatkowe zobowiązanie.

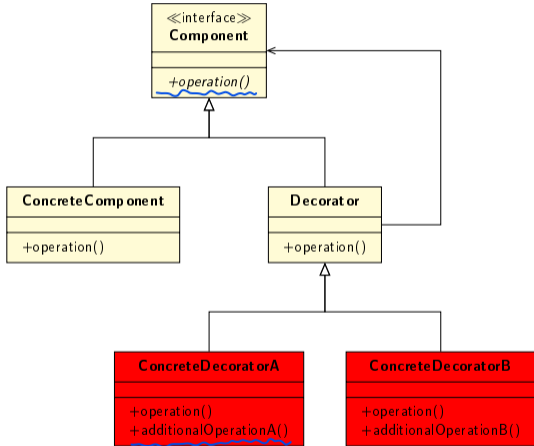
**Konkretny Komponent** zazwyczaj implementuje zdefiniowane z interfejsie operacje w sposób podstawowy. Dodatkowe aspekty tej funkcjonalności definiują **Konkretni Dekoratorzy**.



## Dekorator

- Zarządza odwołaniem do obiektu **Komponent**.
- Definiuje interfejs dopasowany do interfejsu **Komponentu**.

**Dekorator** oddziela **Konkretnych Dekoratorów** od **Konkretnych Komponentów**. Zarządza obiektem komponentu. Może być jego właścicielem lub jedynie użytkownikiem.



## DekoratorKonkretny

- Dodaje zobowiązania do Komponentu.

**Dekorator Konkretny** pozwala na dodanie nowej operacji (metody) do istniejącego już obiektu. Zazwyczaj istnieje wiele różnych dekoratorów i każdy z nich dodaje nowe funkcjonalności. W systemie mogą pojawiać się obiekty "udekorowane" w rozmaity sposób.

**Dekorator** przesyła żądania wykonania operacji do swojego **Komponentu**. **Dekorator** przed przesłaniem żądania lub po może wykonać dodatkowe operacje.

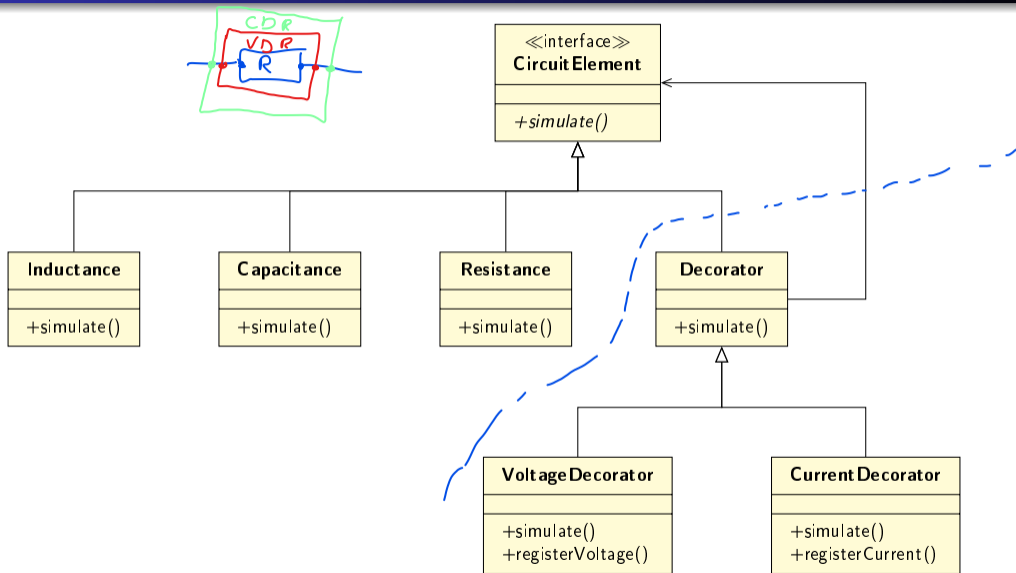


## Obszar zastosowań

Stosuj wzorzec dekoratora, gdy:

- istnieje konieczność dodawania w sposób dynamiczny zobowiązań do istniejących obiektów bez wpływu na ich klientów;
- ze względu na dużą ilość nowych odpowiedzialności stworzenie wszystkich ich kombinacji jest niepraktyczne.

# wzorec projektowy «dekorator»



## Przykładowy problem

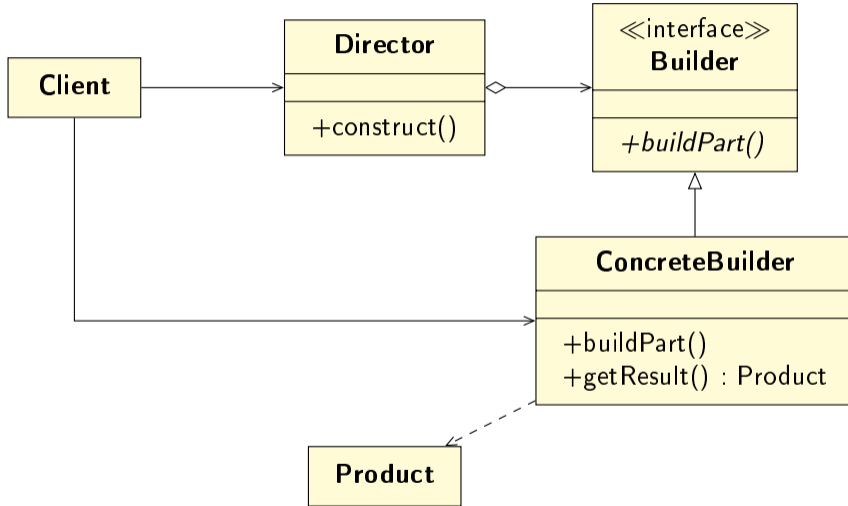
Dany jest symulator układów analogowych wraz ze wszystkimi klasami niezbędnymi do reprezentacji poszczególnych elementów dyskretnych.

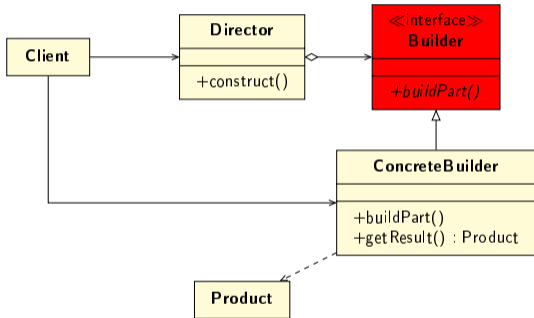
Cechy symulatora:

- Posiada dwa niezależne moduły symulacji : w dziedzinie czasu i częstotliwości.
- Każdy z modułów symulacji posiada własną hierarchię klas odpowiadających symulowanym elementom dyskretnym.
- Istnieje jeden format pliku utrwalającego strukturę symulowanego układu analogowego.

Należy zbudować mechanizm odczytywania struktury pliku, który będzie w stanie budować odpowiednie układy, niezależnie od typu symulacji.

# Wzorzec projektowy «budowniczy»

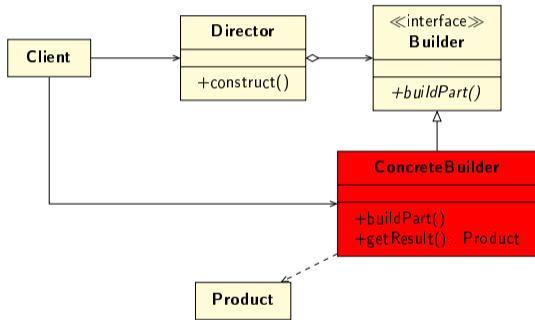




## Budowniczy

- Określa interfejs abstrakcyjny do tworzenia części składowych **Produktu**.

**Budowniczy** definiuje jakich metod mogą używać **Kierownik/cy** i jakie metody muszą być zdefiniowane w konkretnych **Budowniczych**.

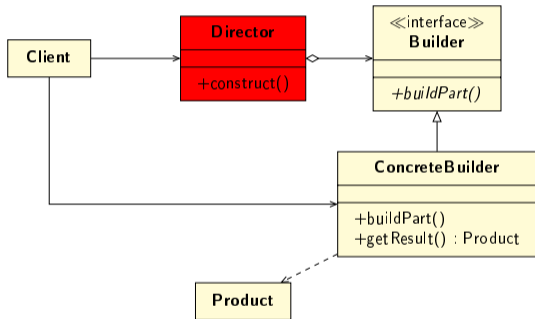


## KonkretnyBudowniczy

- Konstruuje i zestawia ze sobą części **Produktu** przez implementowanie interfejsu **Budowniczego**.
- Definiuje i kontroluje tworzoną przez siebie reprezentację.

Wytwarzany **Produkt** może być inny dla różnych **Budowniczych**. Poszczególne **Produkty** nie muszą należeć do jednej hierarchii. Dom z klocków Lego może powstawać według tych samych ogólnych założeń co dom rodzinny.

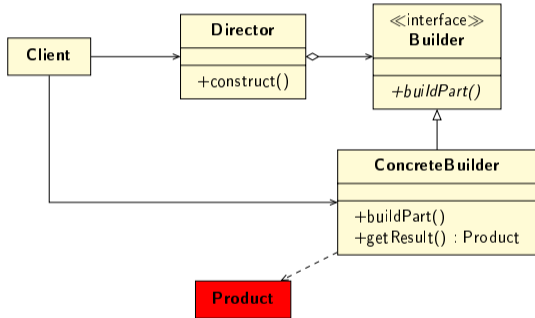
# Wzorzec projektowy «budowniczy»



## Kierownik

- Konstruuje **Produkt**, używając interfejsu **Budowniczego**.

**Kierownik** wie jakie elementy budowanego **Produktu** mają być utworzone i w jakiej kolejności, ale nie musi nic wiedzieć na temat samego **Produktu**. **Kierownik** nie musi wiedzieć jak je łączyć ze sobą oraz jak **Produkt** jest przechowywany. **Kierownik** nie musi wiedzieć z jakim **Budowniczym** współpracuje.



## Produkt

- Reprezentuje konstruowany obiekt złożony.
- Zawiera klasy definiujące części składowe tego obiektu.

Różni **Konkretni Budowniczowie** mogą budować **Produkt** różnego rodzaju.



**Klient** tworzy obiekt **Kierownika** i konfiguruje go konkretnym **Budowniczym**. **Kierownik** definiuje co ma być utworzone i informuje **Budowniczego** o potrzebie zbudowania części **Produktu**.

**Budowniczy** przetwarza żądania **Kierownika** i dodaje niezbędne części do **Produktu**. **Klient** pobiera gotowy **Produkt** od **Budowniczego**.

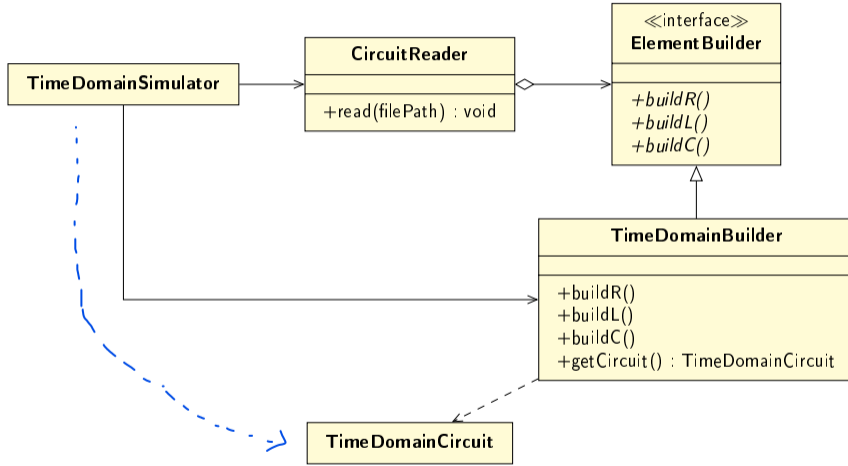
Algorytm tworzenia złożonego **Produktu** jest niezależny od części składowych konkretnego obiektu i sposobu ich tworzenia. Konstruowany **Produkt** może mieć różne reprezentacje w zależności od wybranego **Budowniczego**

## Obszar zastosowań

Stosuj wzorzec budowniczego, gdy:

- dany jest algorytm tworzenia obiektu złożonego z wielu elementów składowych;
- mogą występować różne niepowiązane ze sobą obiekty, które tworzone są z użyciem tego samego algorytmu;
- należy zapewnić jednolity sposób budowania wszystkich obiektów, który jest niezależny od wewnętrznej reprezentacji tych obiektów.

# Wzorzec projektowy «budowniczy»



## Przykładowy problem

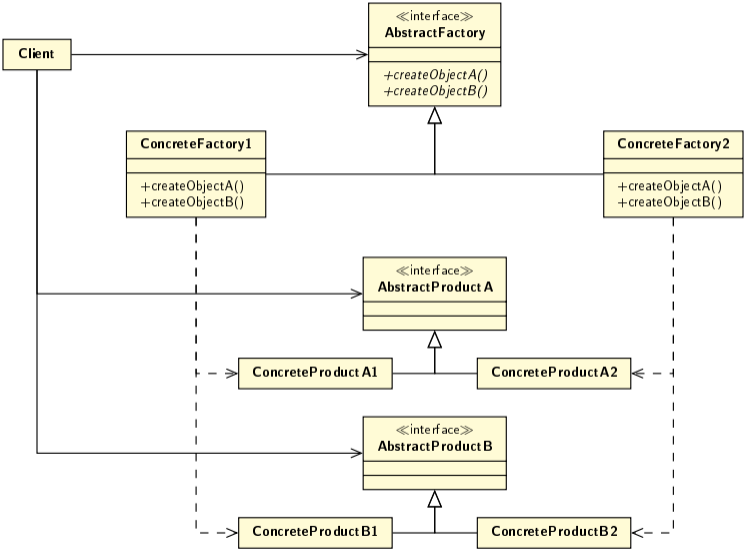
Dany jest symulator układów analogowych wraz ze wszystkimi klasami niezbędnymi do reprezentacji poszczególnych elementów dyskretnych.

Cechy symulatora:

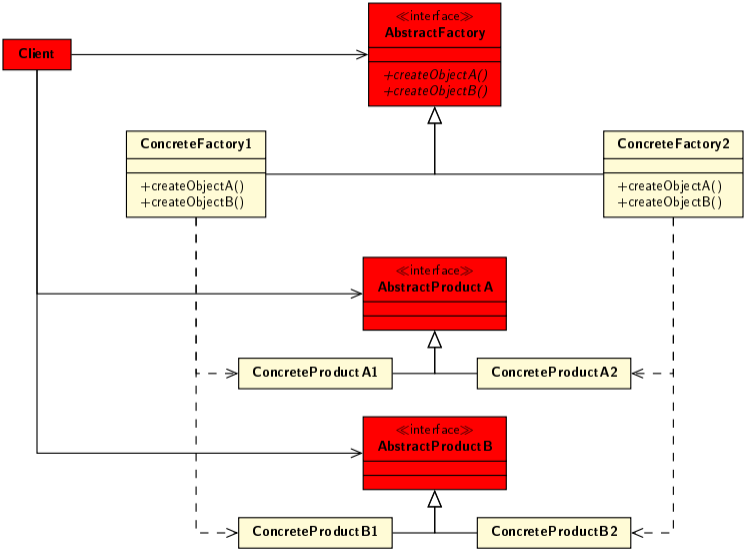
- Poszczególne elementy dyskretne mogą być modelowane w sposób:
  - uproszczony, gdzie elementy cechują się jedynie podstawowymi parametrami – pojemnością, rezystancją oraz indukcyjnością
  - pogłębiony, gdzie oprócz podstawowych parametrów uwzględnione są na przykład pojemność zwojów cewki.
- Ma możliwość przeprowadzania symulacji uproszczonych oraz dokładnych w zależności od rodzaju wykorzystanych elementów.

Należy zbudować mechanizm tworzenia obiektów reprezentujących elementy dyskretne, w zależności od trybu przeprowadzanej symulacji, będzie dostarczał odpowiednie elementy.

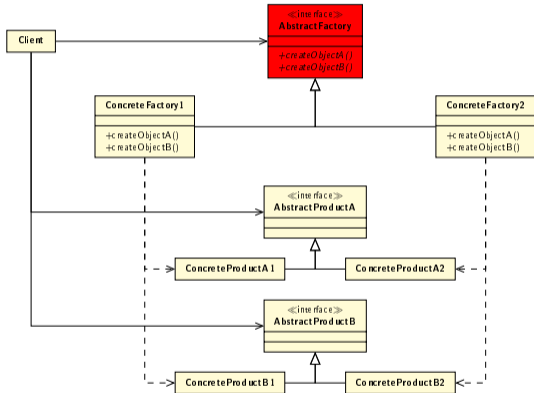
# wzorzec projektowy «fabryka abstrakcyjna»



# wzorec projektowy «fabryka abstrakcyjna»



# wzorzec projektowy «fabryka abstrakcyjna»

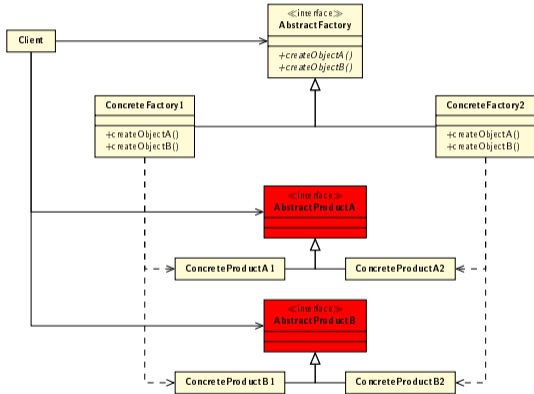


## Fabryka Abstrakcyjna

- Deklaruje interfejs tworzenia **Produktów Abstrakcyjnych**.

**Fabryka Abstrakcyjna** definiuje jakiego typu **Produkty Abstrakcyjne** są tworzonej przez jej konkretne realizacje. Tworzone w ramach fabryki produkty należą do tej samej dziedziny, łączą się w pewien logiczny sposób. **Klienci Fabryki Abstrakcyjnej** nie wiedzą, z którą **Konkretną Fabryką** współpracują. **Fabryka Abstrakcyjna** oddziela **Klientów** od jej konkretnej implementacji.

# wzorec projektowy «fabryka abstrakcyjna»



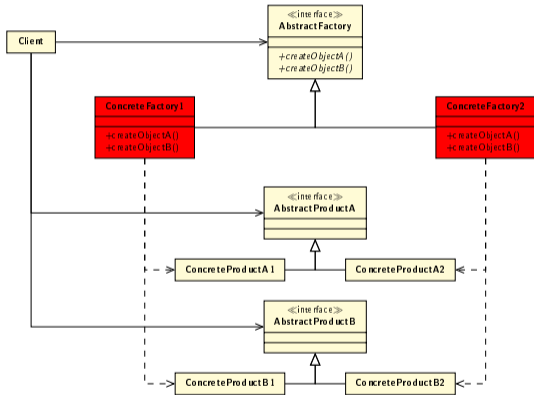
## Produkt Abstrakcyjny

- Deklaruje interfejs dla pewnego rodzaju produktu.

**Produkt Abstrakcyjny** definiuje czym jest obiekt z danej rodziny produktów.



# wzorec projektowy «fabryka abstrakcyjna»

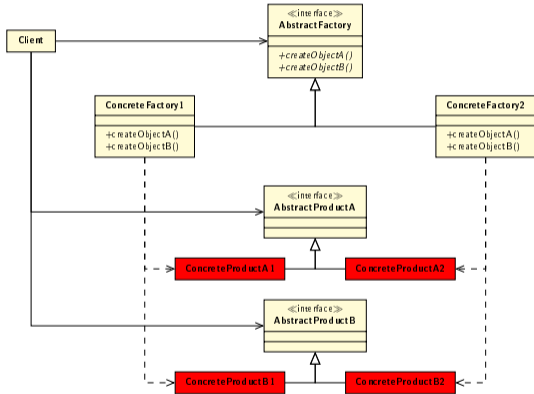


## Konkretna Fabryka

- Implementuje operację tworzenia pewnego rodzaju produktu.

**Konkretna Fabryka** tworzy **Konkretne Produkty**. Sama **Konkretna Fabryka** tworzona jest w oderwaniu od **Klienta**. **Klienci**, którzy używają **Fabryki Abstrakcyjnej** nie wiedzą, z którą **Konkretną Fabryką** współpracują.

# wzorec projektowy «fabryka abstrakcyjna»



## Konkretny Produkt

- Definiuje obiekt będący produktem.
- Implementuje interfejs klasy **Produkt Abstrakcyjny**.

**Konkretne Produkty** tworzone są przez **Konkretne Fabryki**. **Klienci** nie wiedzą, którego konkretnego produktu używają.

**Klienci** korzystają z interfejsu fabryki do tworzenia konkretnych obiektów. **Klienci** nie wiedzą z jaką konkretnie fabryką współpracują, nie wiedzą też jaki mają typ konkretne produkty tworzone przez fabrykę.

**Klienci** zazwyczaj nie mają wpływu na to, z której **KonkretnejFabryki** korzystają. **Klienci** uniezależnieni od konkretnych implementacji obiektów i fabryk skuteczniej mogą implementować swoje odpowiedzialności.

## Obszar zastosowań

Stosuj wzorzec fabryki abstrakcyjnej, gdy:

- w systemie istnieje wiele powiązanych ze sobą rodzin produktów;
- system konfigurowany jest przy użyciu jednej z tych rodzin;
- obiekty z poszczególnych rodzin produktów powinny być używane wspólnie;
- system powinien być izolowany od mechanizmu tworzenia poszczególnych produktów.

# wzorec projektowy «fabryka abstrakcyjna»

