

Wzorce projektowe i dobre praktyki

Maciej Puchała

Agenda

- Czym są wzorce projektowe
- Podział wzorców projektowych
- Podstawowe wzorce kreacyjne
- Podstawowe wzorce strukturalne
- Podstawowe wzorce behawioralne
- Omówienie pozostałych wzorców(jeżeli zostanie nam trochę czasu)
- Clean Code
- PEP8
- Lintery i Autoformatery
- Reguły SOLID



Czym są wzorce projektowe?

Wzorce projektowe to typowe rozwiązania problemów często napotykanym 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 gang of four

Gang of Four – skrót odnoszący się do autorów książki: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software



Podział wzorców

Kreacyjne/Konstrukcyjne

- Fabryka
- Budowniczy
- Prototyp
- Singleton
- Abstrakcyjna Fabryka

Strukturalne

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

Behavioralne/Operacyjne

- Łańcuch zobowiązań
- Polecenie
- Iterator
- Mediator
- Pamiątka
- Metoda szablonowa
- Obserwator
- Stan
- Strategia
- Odwiedzający

Wzorzec Fabryki

Problem:

Wyobraź sobie sytuację, w której prowadzisz sklep internetowy ze znaczkami pocztowymi. Obsługa zamówień odbywa się przez program, który zarządza całym procesem. Program nadzoruje wszystko od złożenia zamówienia do obsługi ewentualnych reklamacji. Jednym z etapów obsługi zamówienia jest wysyłka towaru do klienta.

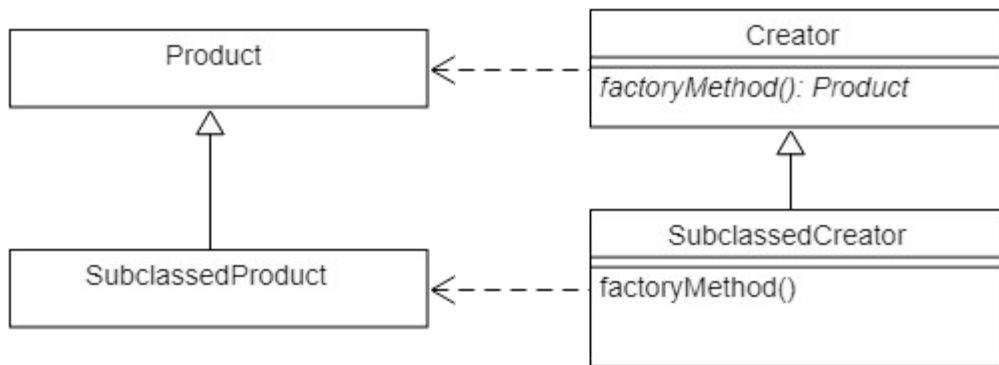
Do tej pory program pozwalał wyłącznie na wysyłkę znaczków używając standardowej poczty. Z biegiem czasu klienci zaczęli oczekiwać dostępności innych sposobów dostawy. Problem polega na tym, że program używa wyłącznie jednego rodzaju wysyłki. Z pomocą w usprawnieniu takiego programu może przyjść metoda wytwórcza (ang. *factory method*).

W tym przypadku metoda wytwórcza może być odpowiedzialna za tworzenie klas odpowiedzialnych za różne rodzaje wysyłek.



Wzorzec Fabryki

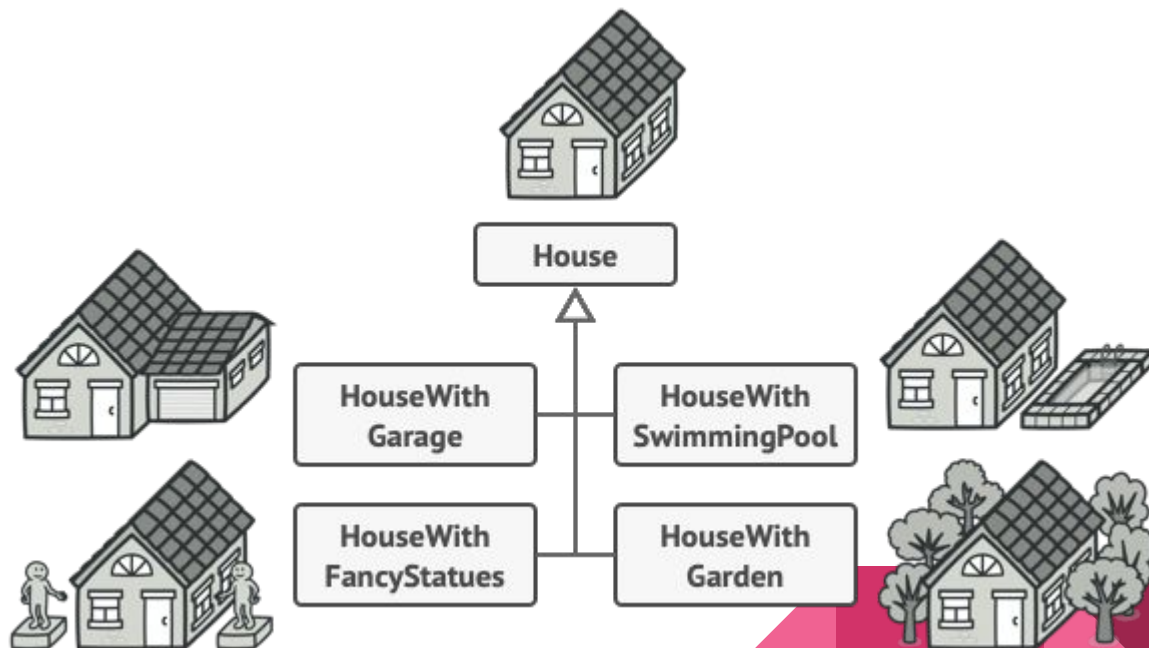
Rozwiązanie:



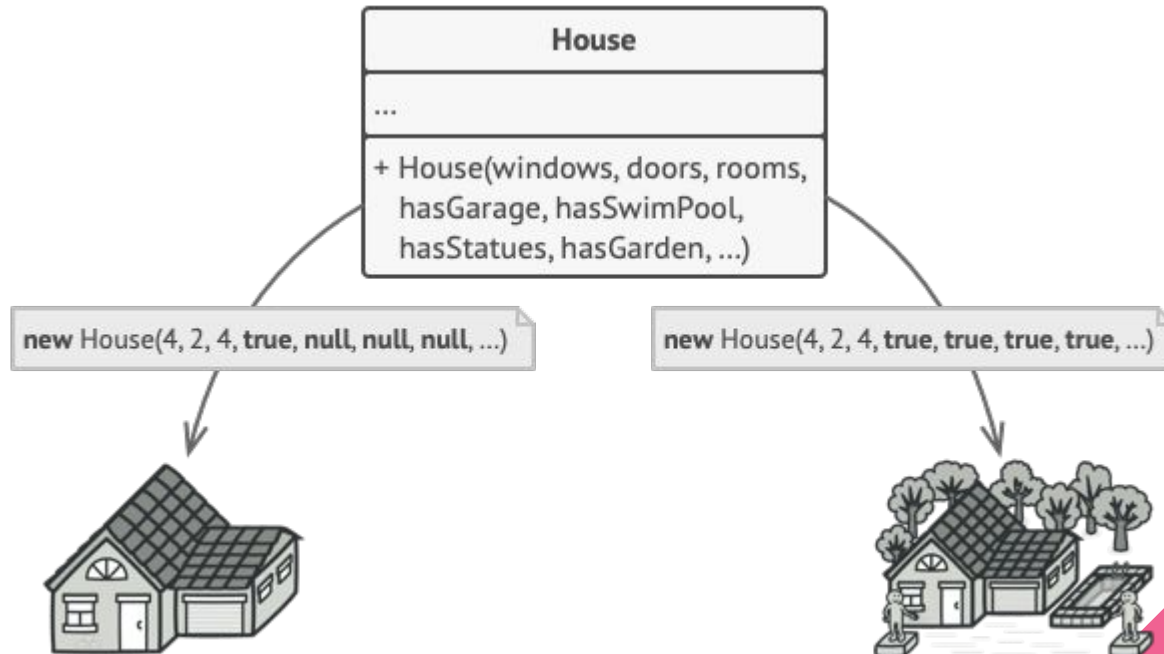
- `Product` – klasa bazowa dla obiektów tworzonych przez metodę wytwórczą,
- `Creator` – klasa zawierająca metodę wytwórczą `factoryMethod`,
- `SubclassedProduct` – przykładowa podklasa `Product`,
- `SubclassedCreator` – podklasa, nadpisująca metodę wytwórczą zwracając instancję `SubclassedProduct`.

Wzorzec Budowniczego

Problem:

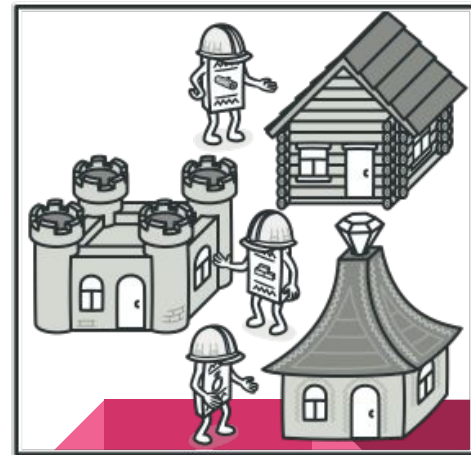
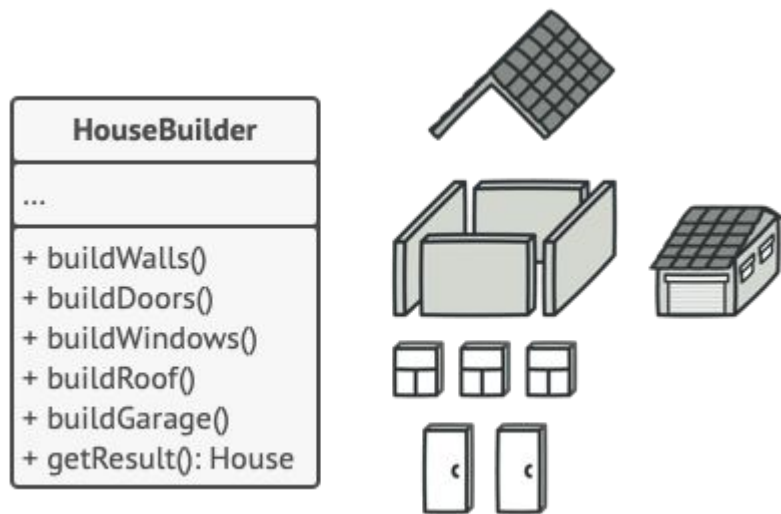


Wzorzec Budowniczego



Wzorzec Budowniczego

Rozwiązanie:




Zadania

Zaimplementuj wzorzec fabryki tworzący magów mag może rzucać zaklęcia zależnie od tego w czym się specjalizuje (Mag ognia, lodu, wody, wiatru etc.)

EASY: Zaimplementuj budowniczego samochodów klasa wysłana na slacku

Harder: Zaimplementuj wzorzec Budowniczego w celu postaci do gry RPG (dla utrudnienia można ograniczyć budowniczego np. ilością punktów atrybutów) niech postać ma atrybuty jak: siła, wytrzymałość, zręczność, inteligencja, charyzma, szczęście oraz swoją klasę która dodaje bonus do konkretnych atrybutów (np. wojownik, mag, kapłan...)



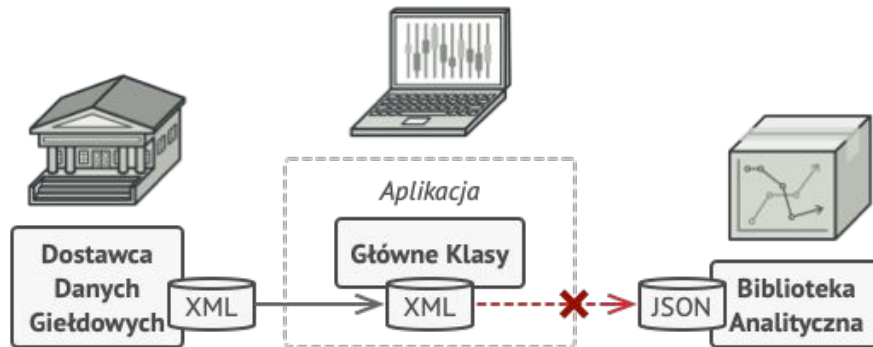
Wzorce strukturalne

Wzorce strukturalne wyjaśniają w jaki sposób można składać obiekty i klasy w większe struktury zachowując przy tym elastyczność i efektywność tych struktur.



Adapter

Problem:

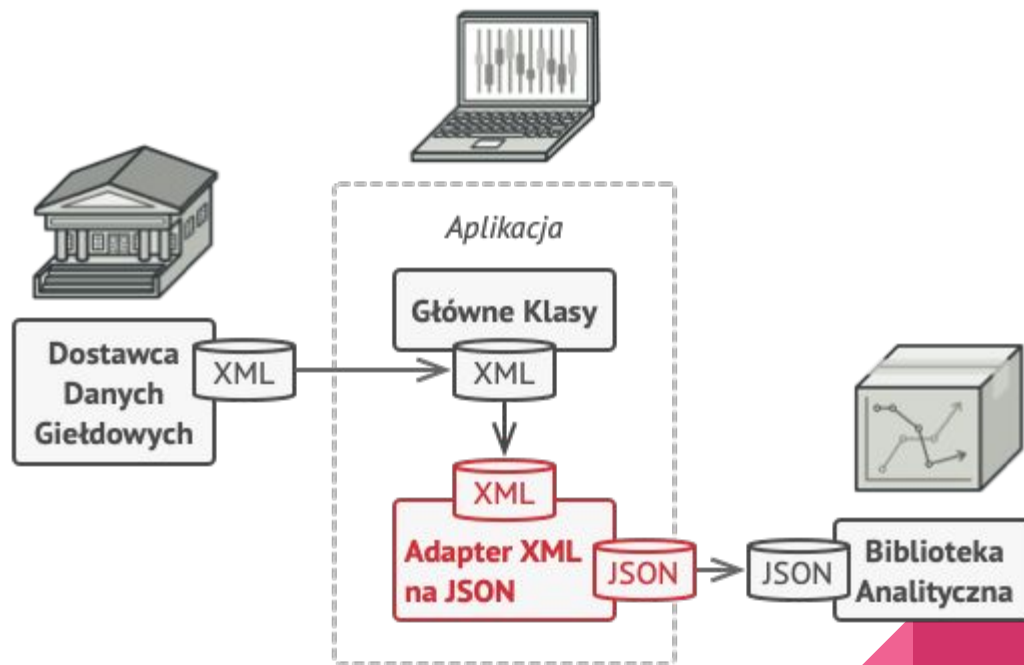


Wyobraź sobie, że tworzysz aplikację monitorującą giełdę. Pobiera ona dane rynkowe z wielu źródeł w formacie XML, a następnie wyświetla ładnie wyglądające wykresy i diagramy.

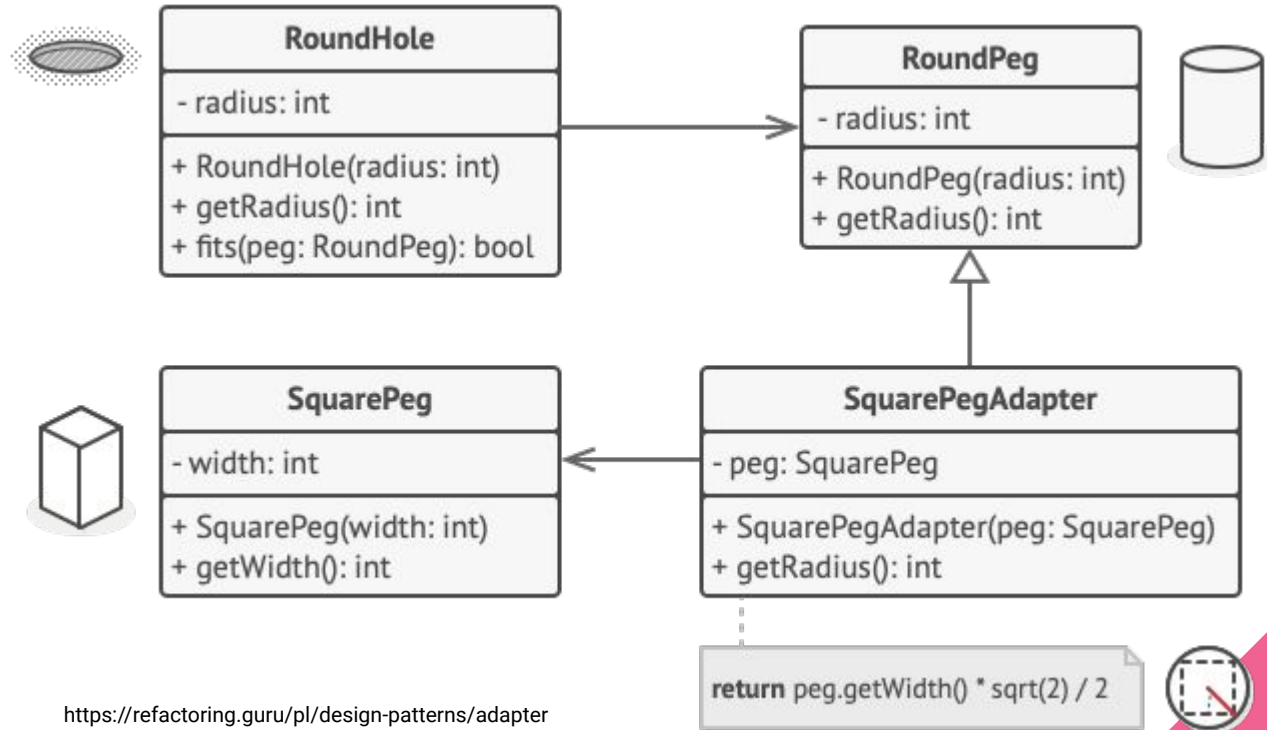
Na jakimś etapie postanawiasz wzbogacić aplikację poprzez dodanie inteligentnej biblioteki analitycznej innego producenta. Ale jest haczyk: biblioteka ta działa wyłącznie z danymi w formacie JSON.

Adapter

Rozwiązanie:



Adapter



Zadanie Adapter

Stwórz adapter dla klasy Motorcycle aby można było wywoływać jak tak samo jak Car.

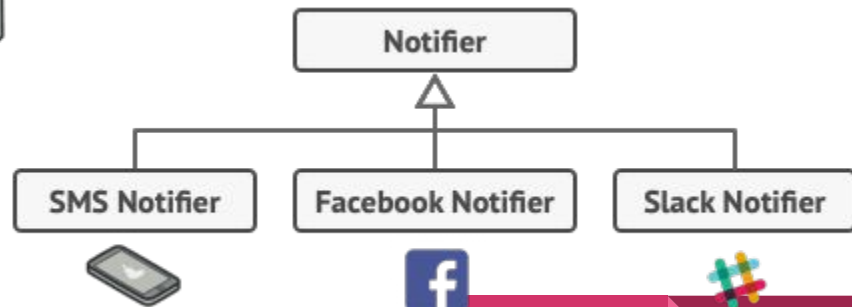
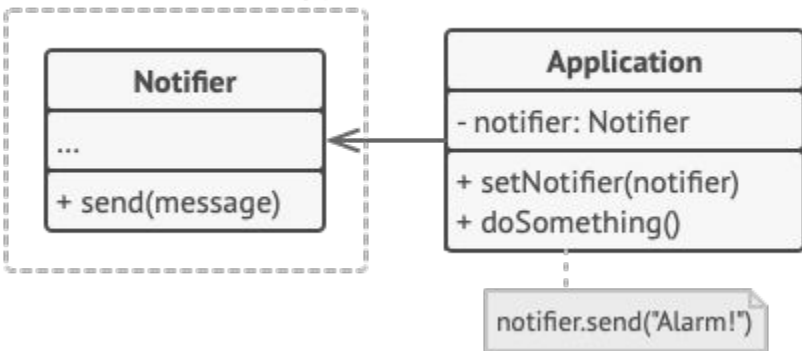
Pomysły na więcej adapterów?



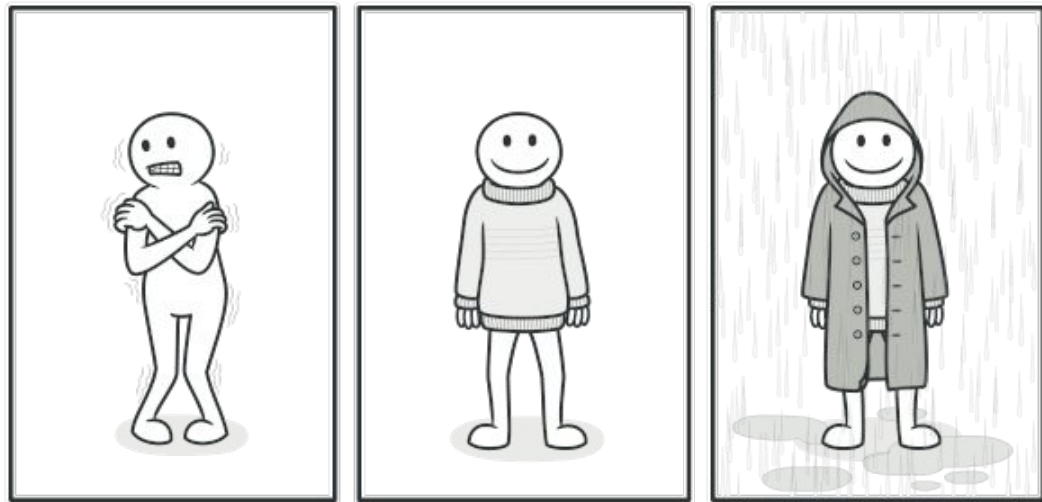
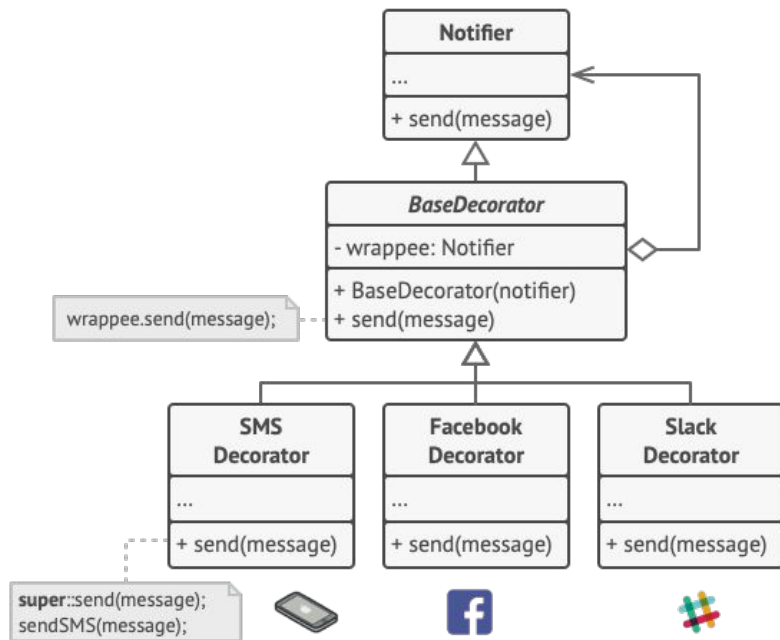
Dekorator

Problem:

Biblioteka Powiadamiająca



Dekorator



<https://refactoring.guru/pl/design-patterns/decorator>

Dekorator zadania

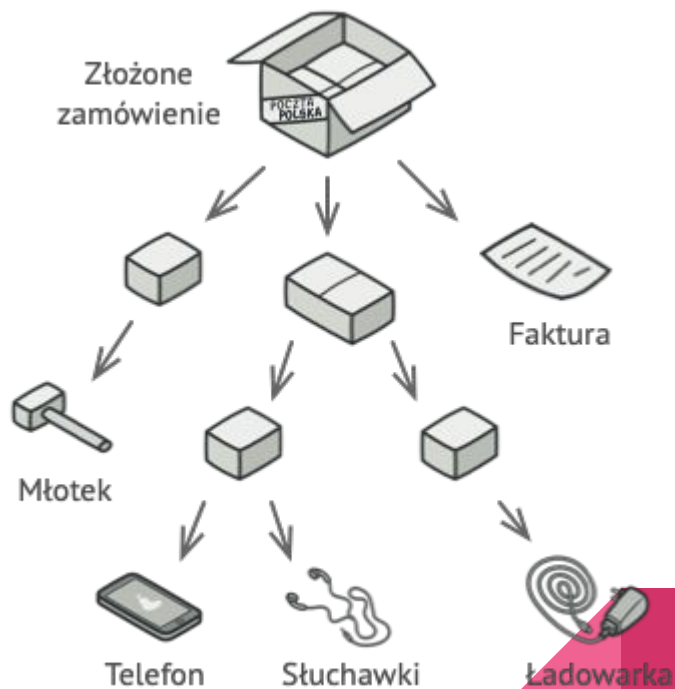
Zaimplementuj wzorzec dekoratora kory pozwoli na dodawanie dodatków do pizzy i będzie potrafił policzyć jej cenę.



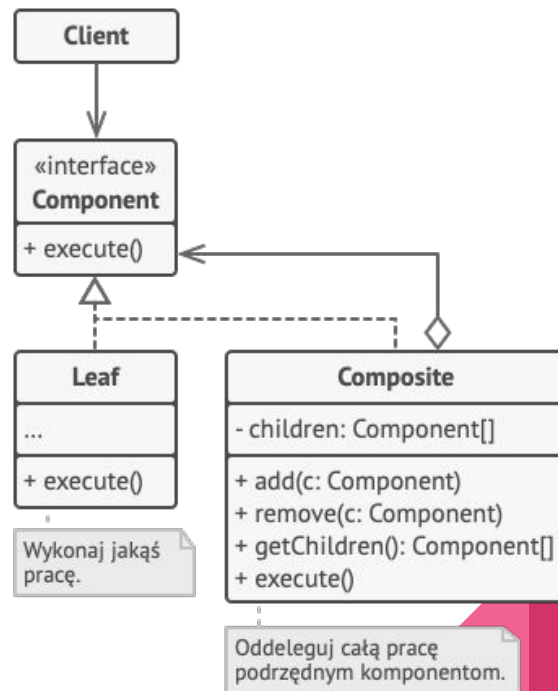
Kompozyt

Problem:

obliczenia wartości paczki



Kompozyt



Zadanie

Zaimplementuj wzorec kompozytu w celu przedstawienia hierarchi w firmie wyglądające w następujący sposób.

- General Manager

 - Manager

 - Developer

 - Developer

 - Junior Dev

 - Manager

 - ...



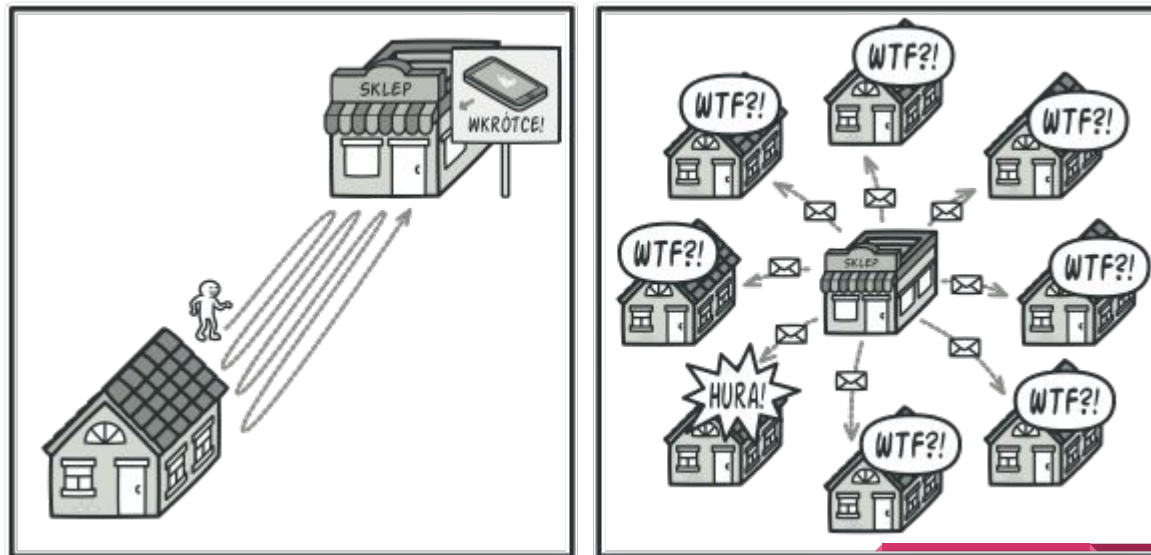
Wzorce behawioralne

Wzorce behawioralne dotyczą algorytmów i rozdzielania odpowiedzialności pomiędzy obiektami.

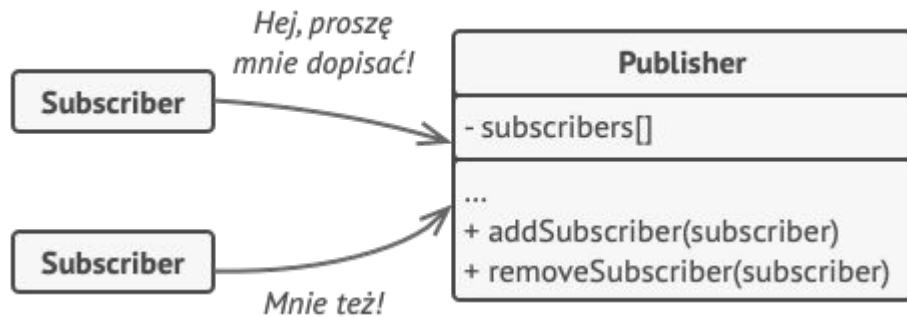


Obserwator

Problem:



Obserwator



Zadania

Zaimplementuj wzorzec obserwatora który pozwoli fanom obserwować poczynania klasy Celebrity.



Reszta wzorców

Przejdziemy do prezentacji SDA oraz przykładów kodu.



Clean Code

"Truth can only be found in one place: the code."

— Robert C. Martin, **Clean Code: A Handbook of Agile Software Craftsmanship**

- Powinien być elegancki - czyli przede wszystkim łatwy do czytania i zrozumienia.
- Powinien być spójny - każda metoda, klasa czy moduł, powinien reprezentować to samo podejście do rozwiązania. Kod nie powinien być rozproszony i "zanieczyszczony" przez zależności i niepotrzebne detale.
- Powinien być zadbany. Programista tworzący czysty kod poświęcił sporo czasu, aby pozostał on uporządkowany i prosty.
- Przechodzi przez wszystkie testy.
- Nie zawiera duplikatów.
- Nie zawiera niepotrzebnych (nieużywanych) klas i metod.

wykłady Uncle Bob'a:

https://www.youtube.com/watch?v=7EmboKQH8IM&list=PLmmYSbUCWJ4x1GO839azG_BBw8rkh-zOj



PEP8 - Style guide for python code

Ogólny zbiór zasad pisania kodu w pythonie.

-PEP8: przedstawia konwencje pisania kodu pełne wytyczne można znaleźć na:

<https://peps.python.org/pep-0008/>

-PEP257: przedstawia dobre praktyki w pisaniu dokumentacji w kodzie tzw. docstringów.

<https://peps.python.org/pep-0257/>



Statyczna analiza kodu

Istnieje wiele narzędzi do analizy kodu pod względem zgodności z standardem PEP8.

- Pylint
- mypy
- flake8

Narzędzia te analizują nasz kod i zwracają informacje o potencjalnych problemach i niezgodnościach ze standardami.



AUTOFORMATERY

Autoformater - analizuje kod pod względem zgodności z PEP 8 i nanosi na niego poprawki:

- black
 - “Bezkompromisowy formater kodu”
 - reguły z PEP8 + dodatki
 - pozwala skupić się na zawartości kodu a nie jego formatowaniu
- yapf - stworzony przez Google
- autopep8



Kawałek nie sformatowanego kodu

Sformatuj kod podany na slacku.

Przejrzyjcie kod który tworzyliście czy to w ramach kursu czy gdzieś osobno dla cwiczenia i sprawdźcie czy linter jak mypy lub flake znajdują jakies błędy i spróbujcie je poprawić.



SOLID

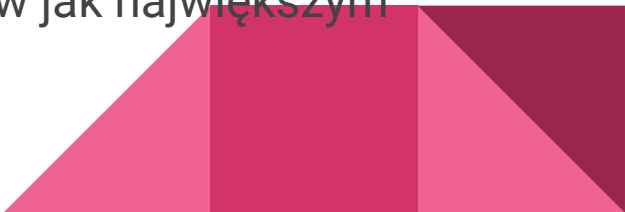
S-Single responsibility - każda klasa powinna być odpowiedzialna za jedną konkretną rzecz.

O-Open/closed- każda klasa powinna być otwarta na rozbudowę ale zamknięta na modyfikacje.

L-Liskov substitution- w miejscu klasy bazowej można użyć dowolnej klasy pochodnej.

I-Interface segregation- interfejsy powinny być konkretne i jak najmniejsze.

D-Dependency inversion- wszystkie zależności powinny w jak największym stopniu zależeć od abstrakcji a nie od konkretnego typu.



S-Single responsibility

Każda klasa powinna być odpowiedzialna za jedną konkretną funkcjonalność.

Ale o co chodzi ?



O- Open/closed

Każda klasa powinna być możliwa do rozszerzenia przez dziedziczenie ale nie powinna być modyfikowana sama w sobie (powinna być zamknięta na modyfikacje).



L - Liskov substitution

Każdy obiekt klasy pochodnej powinniśmy móc podstawić za obiekt rodzica i nie zepsuć programu.

Ale jak to ?



I - Interface segregation

Zasada segregacji interfejsów jest bardzo prosta, mówi aby nie tworzyć interfejsów z metodami, których nie używa klasa. Interfejsy powinny być konkretne i jak najmniejsze.



D - Dependency inversion

Zasada odwrócenia zależności jest prostą i bardzo ważną zasadą. Polega ona na używaniu interfejsu polimorficznego wszędzie tam gdzie jest to możliwe, szczególnie w parametrach funkcji.



Dodatkowe Reguły

KISS - keep it simple and straightforward

DRY - Don't repeat yourself

YAGNI - You aren't gonna need it

