



TRANSITION
TECHNOLOGIES

Design Patterns Wzorce Projektowe

Definicja

- Wzorce projektowe (Design Patterns) są szablonami rozwiązań powtarzających się problemów programistycznych/projektowych
- Są tylko opisem rozwiązania, a nie implementacją
- Zapewniają przejrzyste i optymalne powiązania i zależności pomiędzy klasami oraz obiektami
- Ułatwiają pisanie czystego kodu
- Często przedstawiane w postaci diagramów UML

- **Wzorce projektowe dzielimy na cztery kategorie:**
 - Konstrukcyjne (kreacyjne)
 - Strukturalne
 - Behawioralne (czynnościowe)
 - Architektoniczne

Wzorce konstrukcyjne

- Wzorce konstrukcyjne opisują w jaki sposób obiekty są tworzone
- Są odpowiedzialne za tworzenie, inicjalizację oraz konfigurację obiektów
- Przykłady:
 - Builder (budowniczy)
 - Factory method (metoda wytwórcza)
 - Abstract Factory (fabryka abstrakcyjna)
 - Singleton

Wzorce strukturalne

- Wzorce strukturalne opisują w jaki sposób obiekty są zbudowane
- Struktury powiązanych ze sobą obiektów
- Przykłady:
 - Fasada
 - Adapter (wrapper)
 - Dekorator

Wzorce behawioralne

- Wzorce behawioralne opisują w jaki sposób obiekty się zachowują
- Zachowanie i odpowiedzialność współpracujących ze sobą obiektów
- Przykłady:
 - Obserwator
 - Strategia
 - Łańcuch zobowiązań (Chain of Responsibility)
 - Wizytator

Wzorce architektoniczne

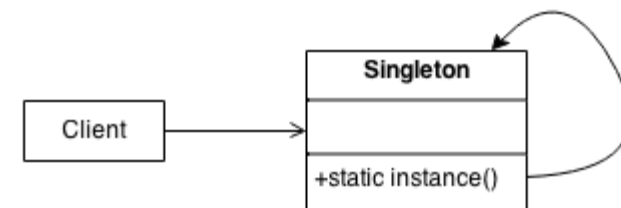
- Wzorce architektoniczne opisują rozwiązania złożonych problemów na wysokim poziomie abstrakcji
- Ogólna struktura systemu informatycznego, elementy z jakich się składa oraz jak poszczególne elementy komunikują się ze sobą
- Przykłady:
 - MVC (Model-View-Controller)
 - P2P (Peer-to-peer)
 - SOA (Service Oriented Architecture)



<https://sourcemaking.com/>

Singleton – wzorzec czy antywzorzec?

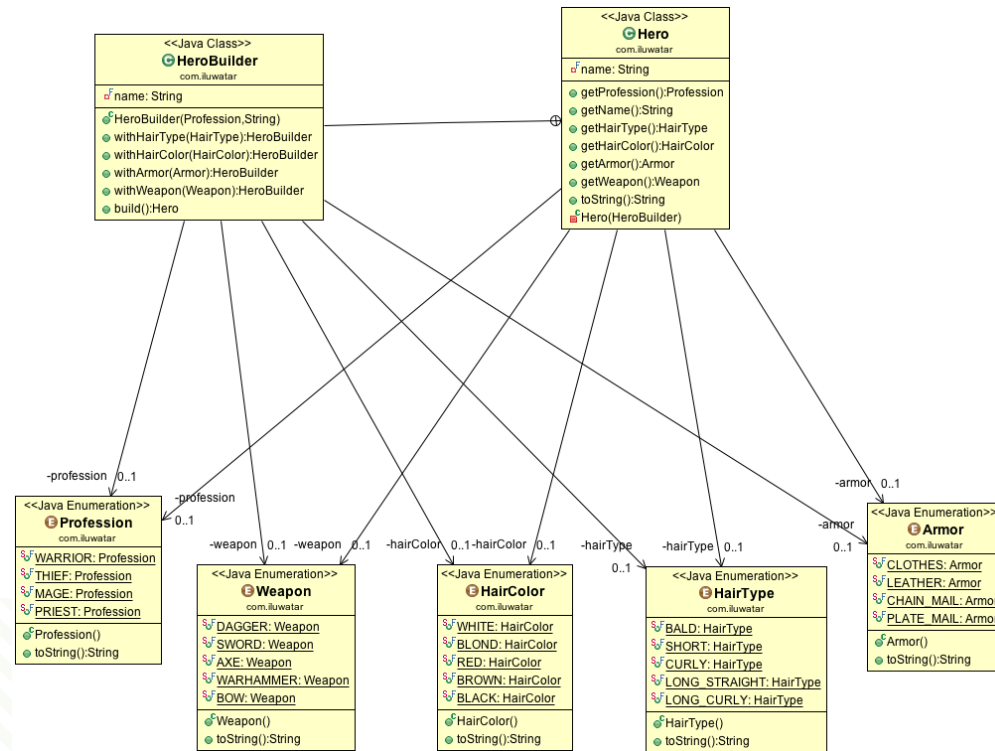
- Cel to ograniczenie ilości tworzonych obiektów danej klasy do tylko jednej instancji
- Przykład: klasa przechowująca konfigurację aplikacji.
 - Z każdego miejsca w systemie możemy ją zmodyfikować i chcemy, żeby zmiany były widoczne również z dowolnego miejsca. Nie możemy pozwolić na to, by w systemie były utrzymywane różne wersje konfiguracji
- Dwa sposoby inicjalizacja obiektu: *eager* vs *lazy*
- Argumenty wskazujące Singleton jako antywzorzec:
 - Trudność testowania
 - Łamanie zasad SOLID
 - Wymaga dodatkowych „zabiegów” gdy mamy aplikacje wielowątkowe
 - Wiele JVMów i tylko jedna instancja?!
 - Nieodpowiedzialne zabawy z refleksją
- Przerzucenie odpowiedzialności na Javę: **Enum**



Więcej info: <https://stormit.pl/singleton>

Builder – rozwiązanie na wieloargumentowe konstruktory

- Stosowany do konstruowania obiektów poprzez wcześniejsze stworzenie jego fragmentów. Składamy od szczegółu do ogółu (np. budowanie domu). Obiekty mogą być rozmaitych postaci, a wszystko opiera się na jednym procesie konstrukcyjnym.
- W konkretnych budowniczych decydujemy o tym, jak dany obiekt jest tworzony

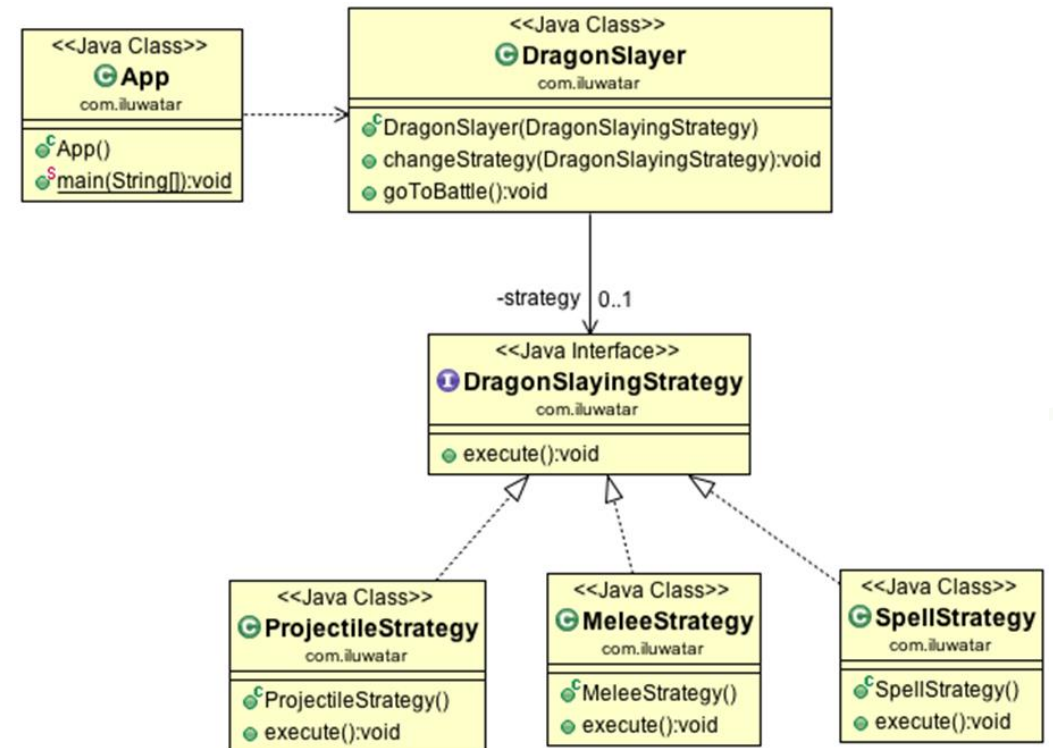


Wzorzec fabryki to SOLIDny przyjaciel

- Używamy tam, gdzie chcemy odciąć się od tworzenia instancji klas posługując się konkretnym typem
- Łatwy sposób wspiera re-używalność kodu (procesu inicjalizacji danej rodziny klas) w innym miejscu systemu
- Skupienie logikę w metodzie fabrykującej, dzięki czemu zmiany w kodzie można wprowadzić w jednym miejscu systemu
- **Wspiera hermetyzację, która jest filarem OOP**
 - Dostarcza dodatkową warstwę abstrakcji enkapsulując odpowiednią logikę wewnątrz fabryki
 - Upraszcza kod
 - Unikamy powtarzalności
 - Ukrywamy logikę
- **Rodzaje fabryk:**
 - Prosta fabryka (simple factory)
 - Fabryka statyczna (static factory)
 - Metoda fabrykująca (factory method)
 - Fabryka abstrakcyjna (abstract factory)

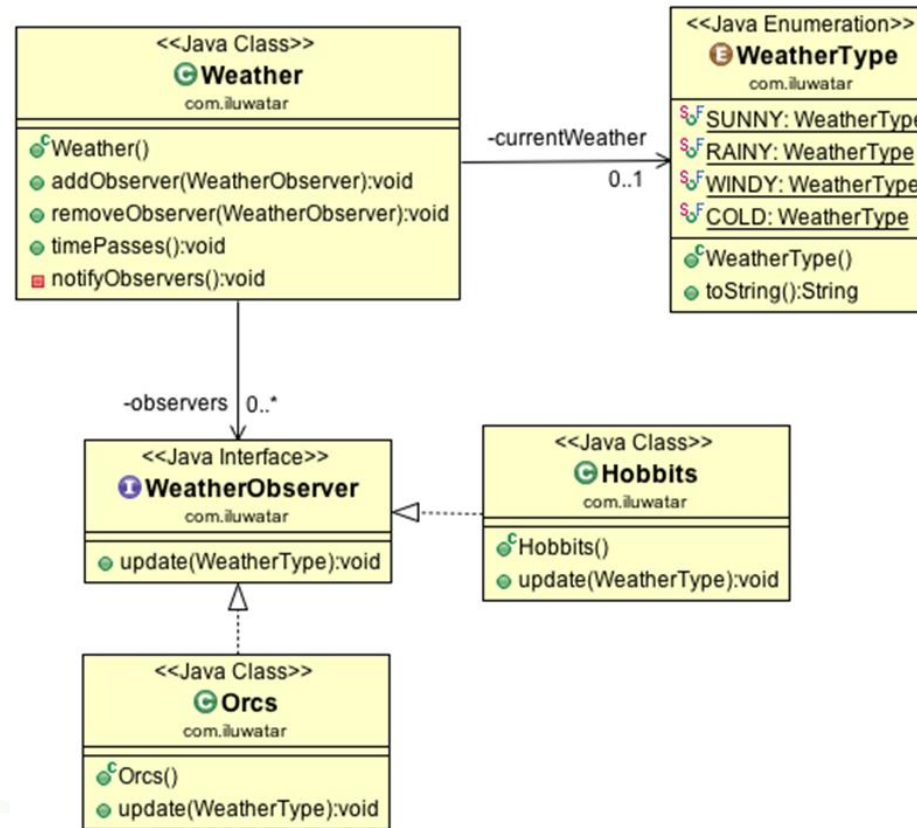
Strategia

- Obsłużenie sytuacji, w której kontekst zmienia się dynamicznie
- Jest pewien problem i można dojść do jego rozwiązania na kilka sposobów (podjęcie różnych strategii)
- Wpływ na sposób rozwiązania danego problemu mogą mieć np. parametry wejściowe
- **Przykład:**
 - rodziny algorytmów znajdujące dany element w kolekcji
 - pokonanie głównego bossa w grze



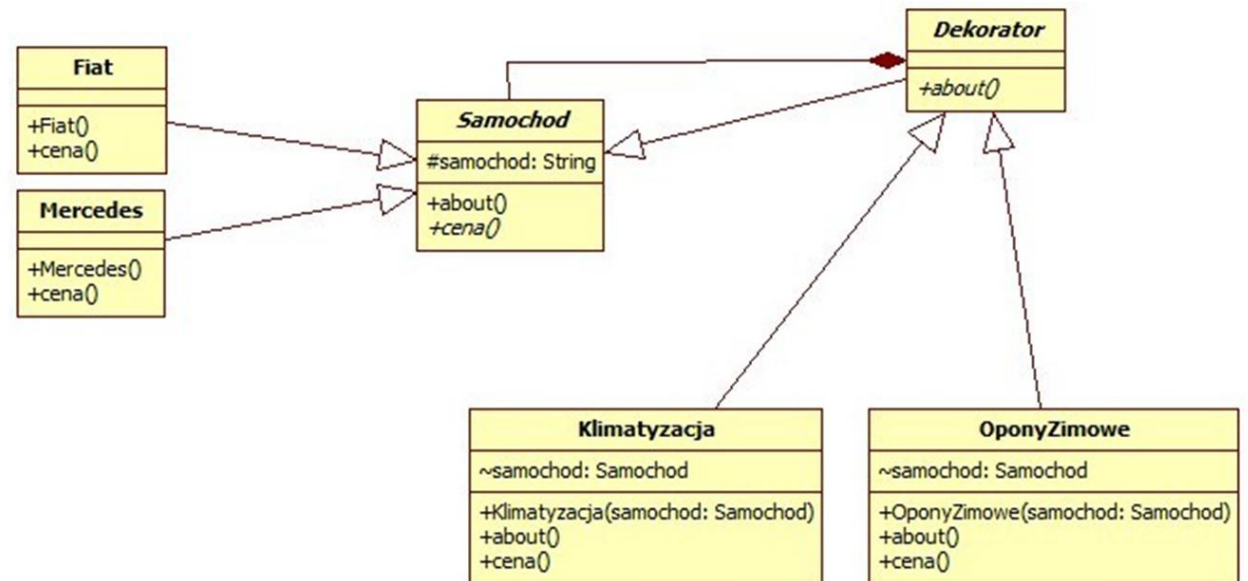
Obserwator

- Obrazuje zależność jeden-do-wielu
- Kiedy zmienia się stan jednego obiektu, zależne od niego obiekty zostają o tym powiadomione i również zmienia się ich stan



Dekorator

- Uważany jako alternatywa dla dziedziczenia, gdyż tak samo jak ono rozszerza funkcjonalności klasy podstawowej
- Idea tego wzorca opiera się na mechanizmach kompozycji oraz delegacji. Obiekt dekorujący zawiera obiekt dekorowany (kompozycja), natomiast dekorator deleguje wywołanie wybranej metody do kolejnego dekoratora lub do metody pochodzącej z klasy dekorowanej, po drodze dodając „swoją” funkcjonalność



Adapter (Wrapper)

- Konwertuje interfejs, z którego korzysta dana klasa, w inny interfejs, który jest oczekiwany
- Pozwala klasom współpracować, ze sobą, pomimo korzystania z różnych interfejsów
- Służy do przystosowania interfejsów obiektowych, tak aby możliwa była współpraca obiektów o niezgodnych interfejsach

