

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

PODZIAŁ WZORCÓW

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

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
 - Prototyp
 - Object pool

WZORCE STRUKTURALNE

- Opisują w jaki sposób obiekty są zbudowane
- Struktury powiązanych ze sobą obiektów
- Przykłady:
 - Fasada
 - Adapter (wrapper)
 - Dekorator
 - Bridge
 - Proxy

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
 - Polecenie (Command)
 - Iterator
 - Null Object
 - Metoda szablonowa (Template)

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)

GDZIE SZUKAĆ INFORMACJI I PRZYKŁADÓW

■ Literatura

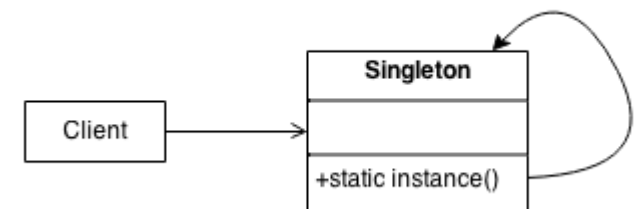
- *Head First Design Patterns* - Elisabeth Freeman & Kathy Sierra
- *Design Patterns: Elements of Reusable Object-Oriented Software* - Erich Gamma, John Vlissides, Ralph Johnson & Richard Helm (Gang of Four)

■ Internet

- <https://sourcemaking.com/>
- <https://refactoring.guru/design-patterns>
- <https://www.youtube.com/playlist?list=PLrhzvlcii6GNjpARdnO4ueTUAVR9eMBpc> (Christopher Okhravi)

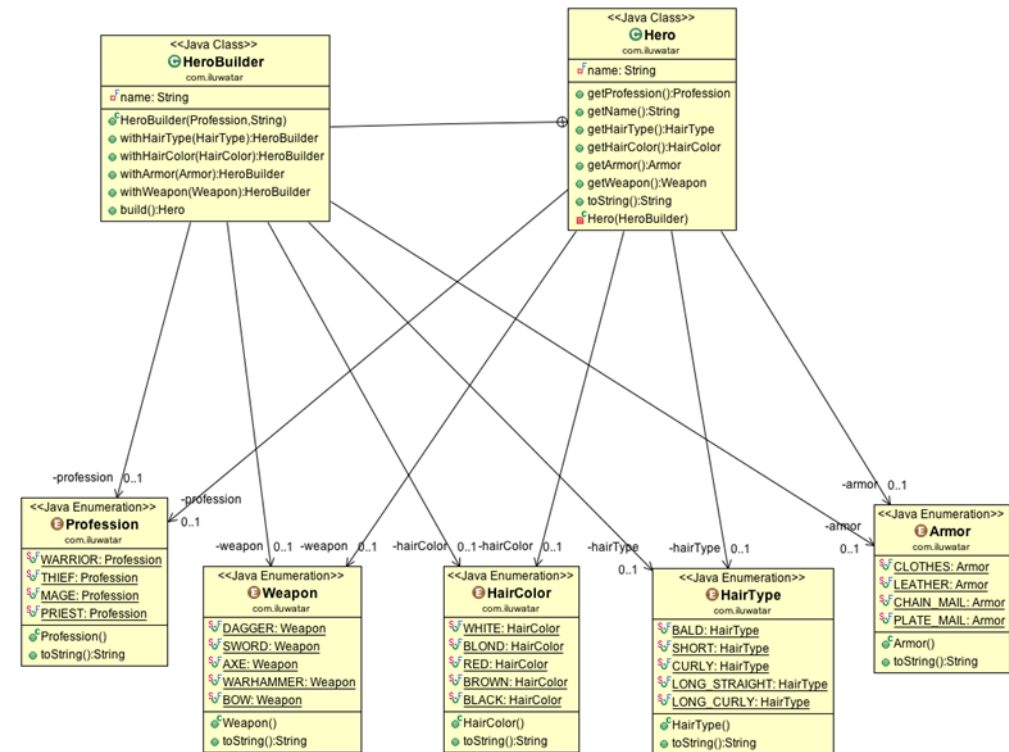
SINGLETON

- 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
 - `java.lang.Runtime#getRuntime()`
 - `java.lang.System#getSecurityManager()`



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
- Java
 - `java.lang.StringBuilder#append()` (unsynchronized)
 - `java.nio.ByteBuffer#put()` (also on `CharBuffer`, `ShortBuffer`, `IntBuffer`, `LongBuffer`, `FloatBuffer` and `DoubleBuffer`)
 - `java.util.stream.Stream.Builder`

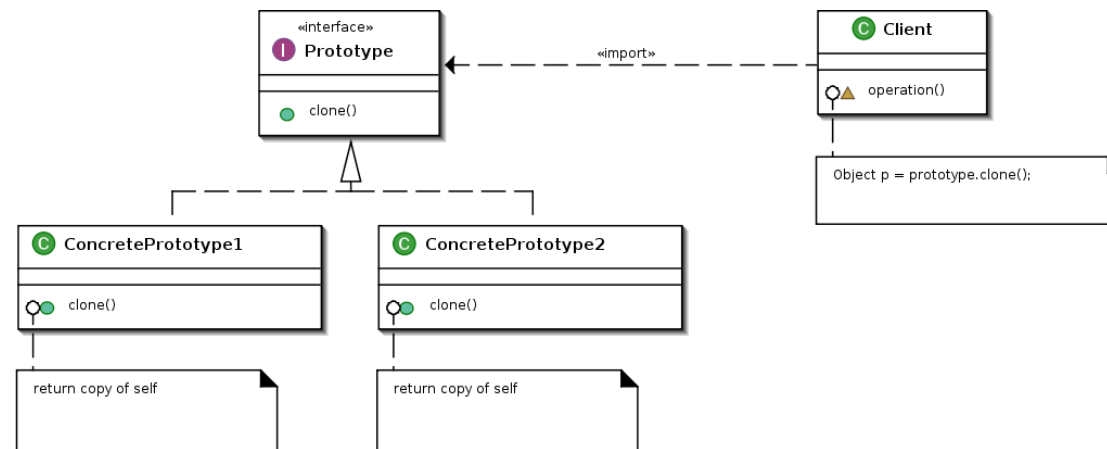


FABRYKA

- 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)
- `javax.xml.parsers.DocumentBuilderFactory#newInstance()`
 - `javax.xml.transform.TransformerFactory#newInstance()`
 - `javax.xml.xpath.XPathFactory#newInstance()`
 - `java.util.Calendar#getInstance()`
 - `java.util.ResourceBundle#getBundle()`
 - `java.text.NumberFormat#getInstance()`
 - `java.nio.charset.Charset#forName()`

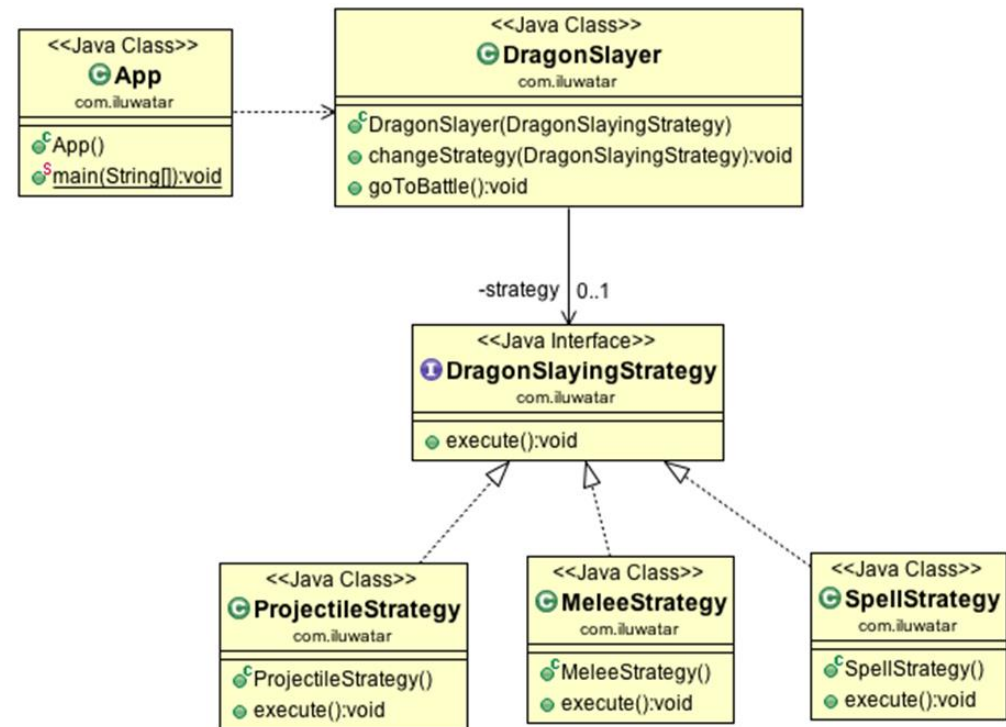
PROTOTYP

- Umożliwia tworzenie obiektów danej klasy z wykorzystaniem już istniejącego obiektu (prototypu)
- Głównym celem jest wyznaczenie sposobu (-ów) w jaki tworzone są obiekty
- Możemy zaimplementować metodę klonującą nasz obiekt, tj. tworzymy prawdziwą kopię instancji naszego obiektu w czasie wykonywania



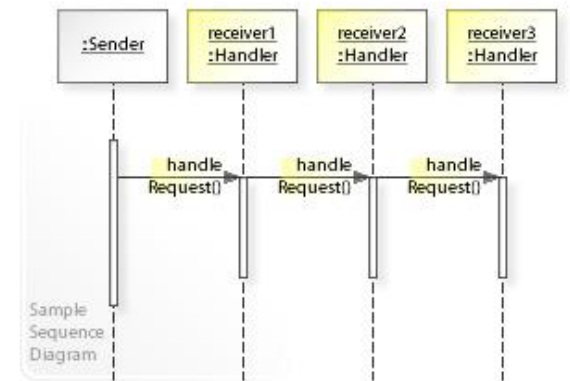
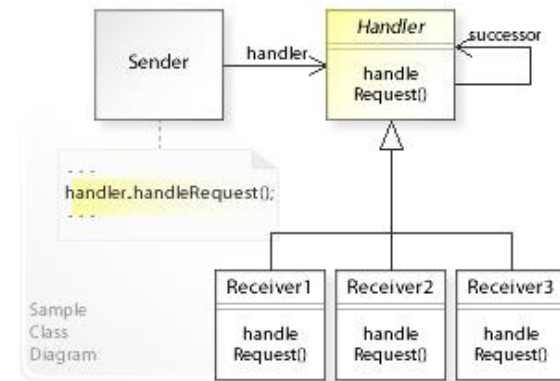
STRATEGIA

- 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
- Java
 - `java.util.Comparator#compare()`
 - `javax.servlet.http.HttpServlet`
 - `javax.servlet.Filter#doFilter()`



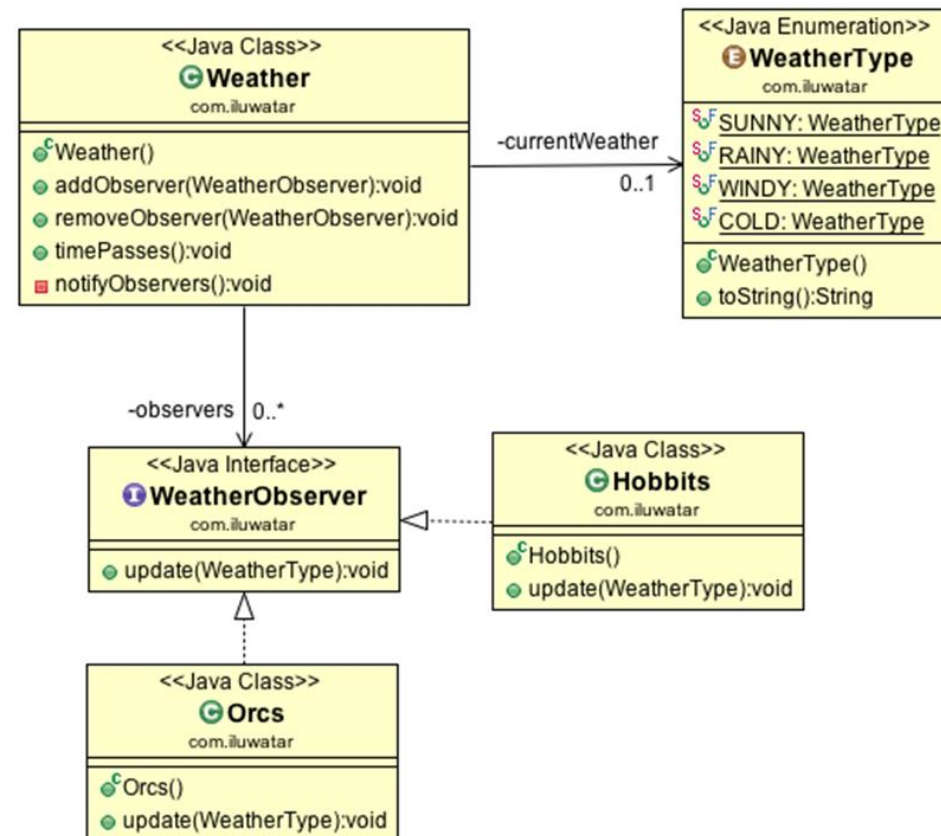
ŁAŃCUCH ZOBOWIĄZAŃ

- Zestaw ściśle powiązanych ze sobą obiektów
- Każdy obiekt ma przypisane określone zadanie oraz wskazanie na następny obiekt, który ma wykonać kolejne (inne) zadanie
- Wzorzec używany przy implementacji komunikacji wysyłający-otrzymujący
- Java
 - `java.util.logging.Logger#log()`
 - `javax.servlet.Filter#doFilter()`



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
- Java
 - `java.util.Observer`/`java.util.Observable`
 - Implementacje `java.util.EventListener`

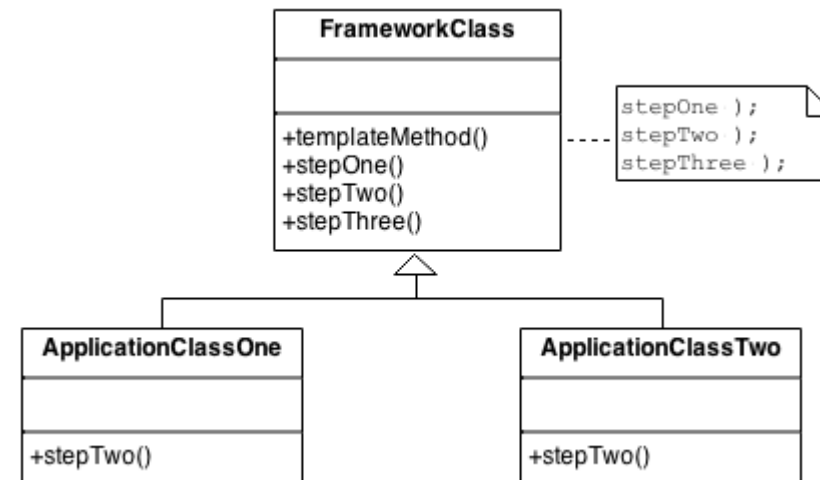


METODA SZABLONOWA (TEMPLATE)

- Umożliwia definiowania szkieletu algorytmu w klasie bazowej i umożliwieniu podklasom nadpisanie kolejnych kroków bez zmieniania całkowitej struktury algorytmu

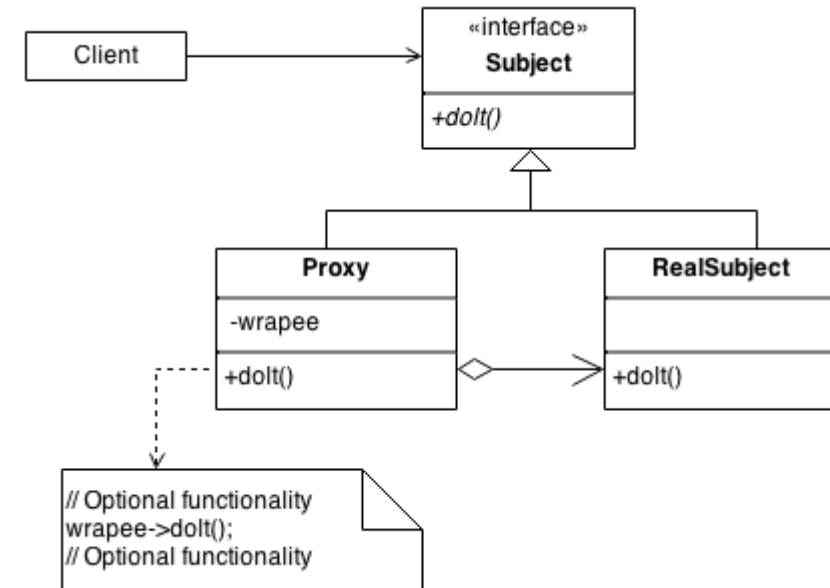
- Java

- `java.util.ArrayList`
- `java.util.AbstractSet`
- `java.util.AbstractMap`



PROXY

- Dostarcza klasę która jest pomostem (interfejsem) do innego obiektu
- Ma to na celu podniesienie wydajności i obniżenie „kosztów” dostępu
- Zbliżone działanie do tzw. „cache”
- Java
 - `java.lang.reflect.Proxy`
 - `javax.persistence.PersistenceContext`



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
- Java
 - `java.util.Arrays#asList()`
 - `java.util.Collections#list()`
 - `java.util.Collections#enumeration()`
 - `java.io.InputStreamReader(InputStream)`

