Design Patterns

Autor:

Kilka słow o mnie

Software Developer @ Onegini

Główny język programowania: JAVA

Dodatkowo: JavaScript, Python, C++

5+ lat na rynku IT

Wejście na rynek IT:

- Podobna ścieżka jak Wasza!
- Studia nietechniczne
- Kursy

Kontakt: bojanowski.michal89@gmail.com

Autor: Michał Bojanowski

WZORCE PROJEKTOWE - WPROWADZENIE

WZORZEC PROJEKTOWY

Jest to sprawdzony, uniwersalny sposób na rozwiązanie **powtarzających** się problemów projektowych w procesie wytwarzania oprogramowania.

Autor: Michał Bojanowski

PODZIAŁ NA GRUPY

- CREATIONAL (konstrukcyjne) - grupa wzorców opisująca jak **stworzyć** dany obiekt
- STRUCTURAL - grupa wzorców opisująca **struktury powiązanych** ze sobą obiektów
- BEHAVIORAL (operacyjne) - grupa wzorców opisująca **zachowanie** i odpowiedzialność powiązanych ze sobą obiektów

Autor: Michał Bojanowski

CREATIONAL

- Singleton
- Builder
- Prototype
- AbstractFactory
- Factory Method

Autor: Michał Bojanowski

STRUCTURAL

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Autor: Michał Bojanowski

Behavioral

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Autor: Michał Bojanowski

Singleton

Problem?

- Stworzenie klasy, która w danej aplikacji może być stworzona tylko **RAZ**

Każda klasa która chce skorzystać z singletonu korzysta z dokładnie tej samej **instancji**.

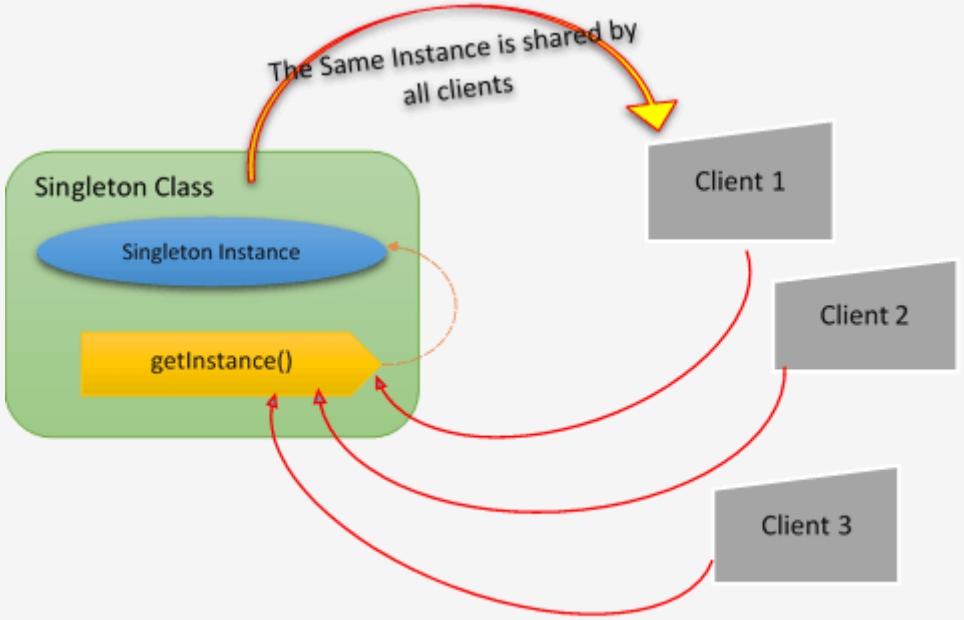


Figure 1: Singleton Design Pattern Overview

Singleton

Sposoby tworzenia:

- Lazy - instancja tworzona podczas pierwszej próby wyłuskania
- Eager - instancja tworzona na starcie aplikacji
- Double-checked (thread safe) - instancja zapewniająca bezpieczny dostęp z wielu wątków
- Enum - instancja stworzona za pomocą ’enum’

Autor: Michał Bojanowski

Singleton - plusy i minusy

Plusy:

- oszczędność pamięci (1 instancja vs wiele instancji)

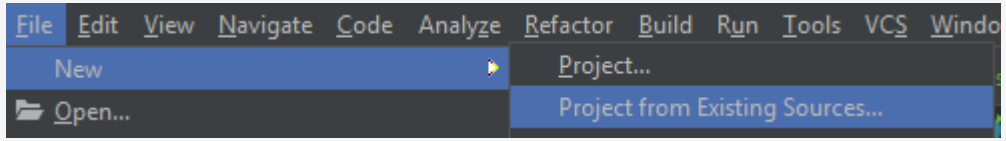
- zysk jeżeli tworzenie obiektu jest kosztowne (np. w kontekscie czasu)

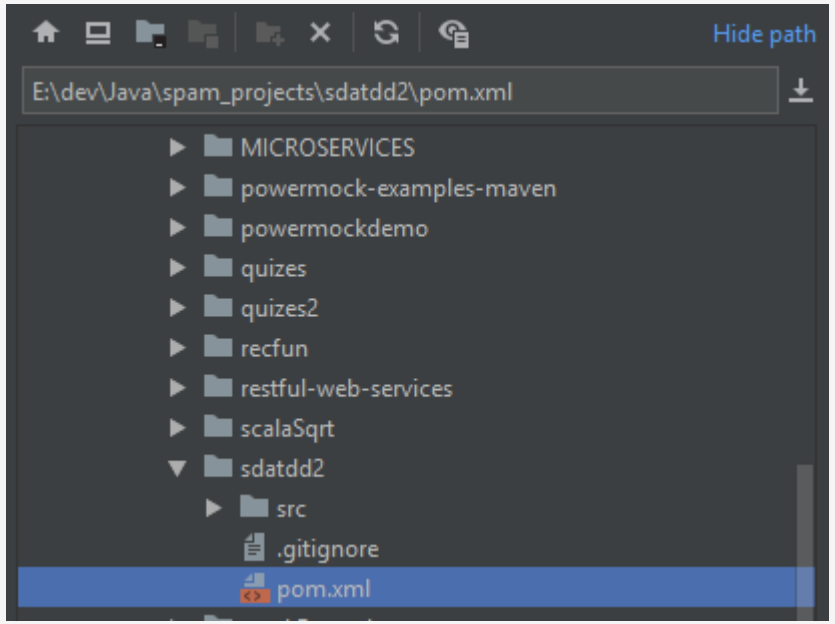
Minusy:

- klasyczny singleton powoduje stworzenie dodatkowych zależności w testach jednostkowych
- źle użyty singleton tworzy wiele ukrytych zależności...
- ... co może zmniejszać czytelność kodu (kto, kiedy i gdzie zmienia zawartość singletonu?)

Autor: Michał Bojanowski

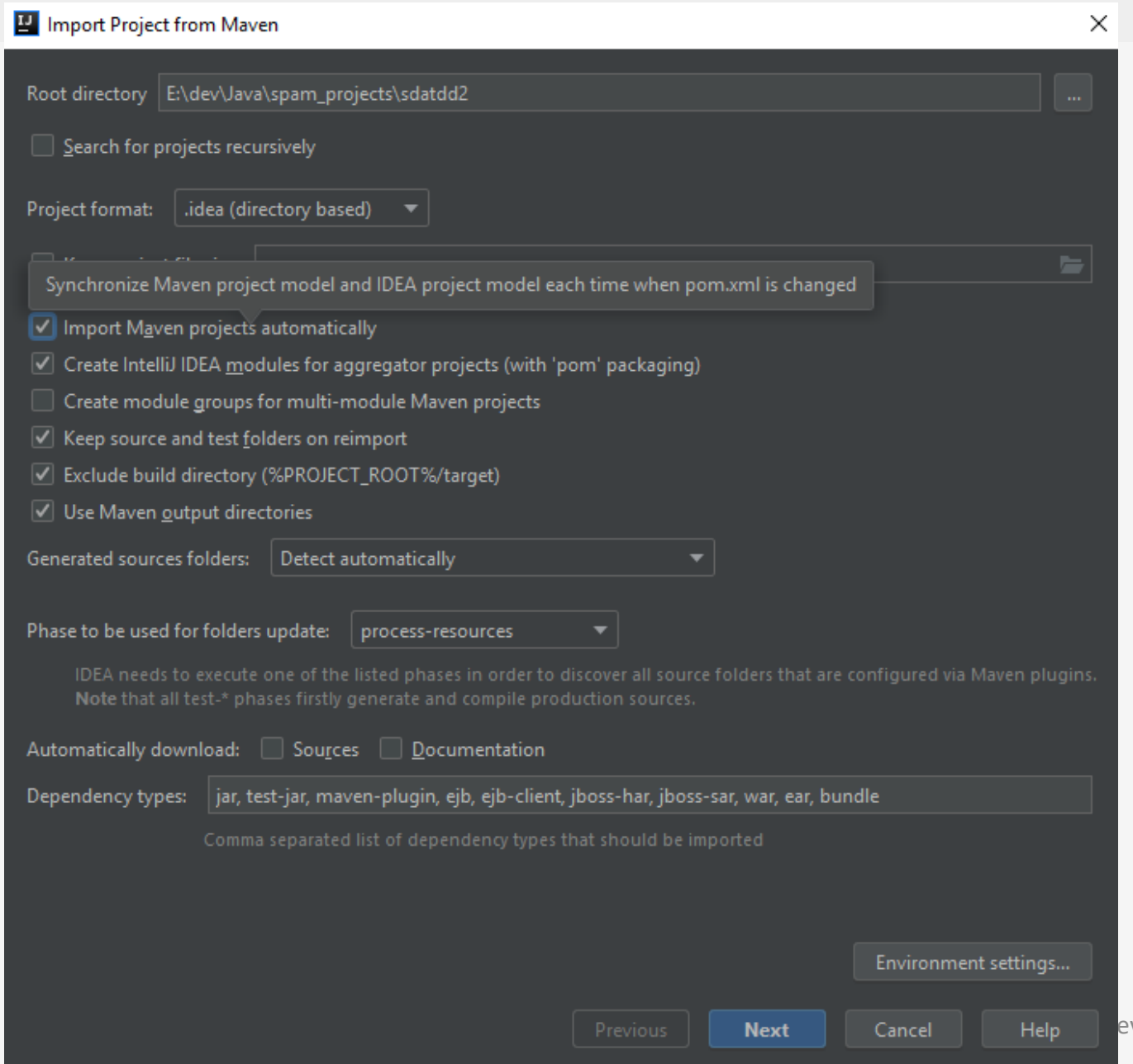
IDEA – poprawne importowanie projektów



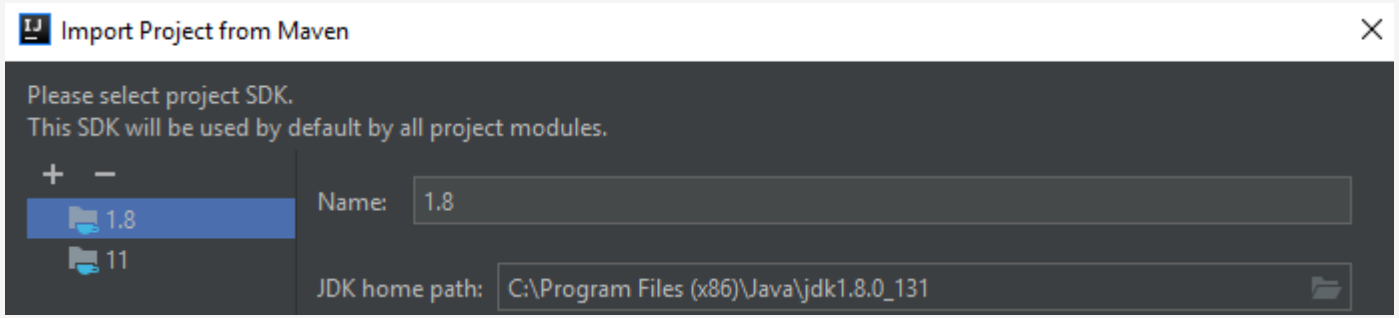


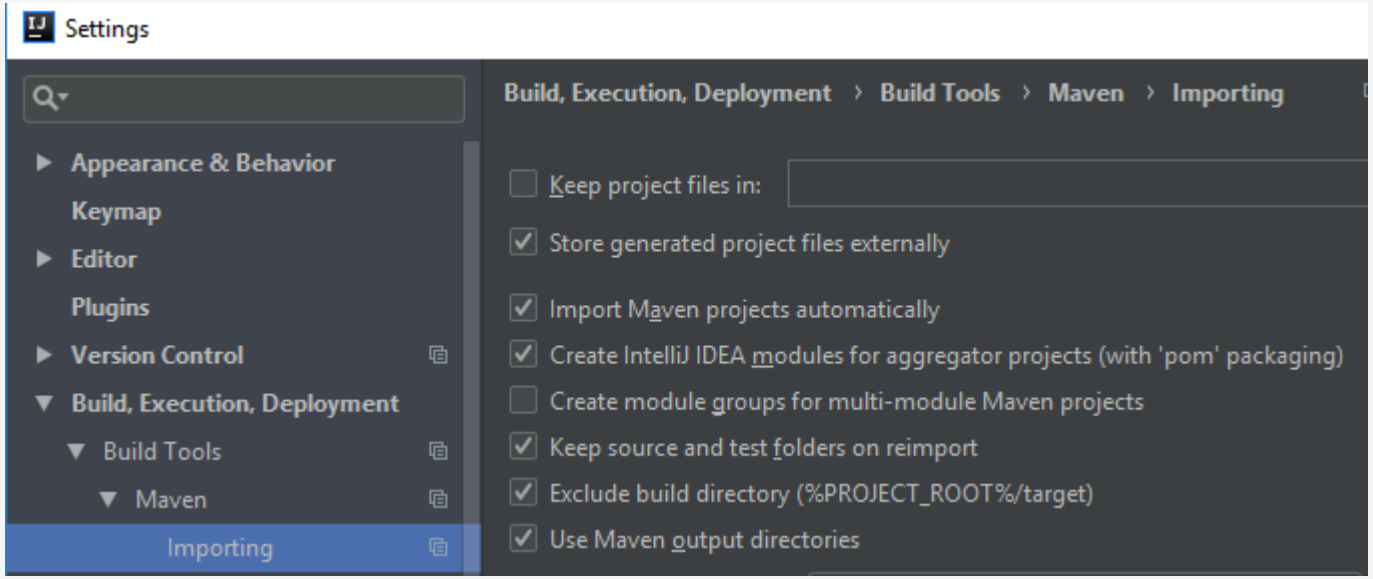
Autor: Michał Bojanowski

IDEA – poprawne importowanie projektów



IDEA – poprawne importowanie projektów





Autor: Michał Bojanowski

IDEA + maven – wymuszenie reimportu

CTRL + SHIFT + A

Autor: Michał Bojanowski

Ćwiczenie 1 (pl.sdacademy.designpatterns.signleton.eager)

1. Zaimportuj projekt https://github.com/Mbojanow/sdadesignpatterns
2. Stwórz singleton (typu EAGER) Counter, który przechowuje liczbę. Jej początkowa wartość jest równa 0. Dodaj 3 metody do klasy Counter
   - metodę która zwiększa licznik odpowiednio o 1, 2 i 3
   - w każdej z tych metod po zwiększeniu wartości wyświetl ją na ekran
3. W metodzie main pobierz dwie referencji singletonu. Wywołaj kilka razy metody na stworzonych referencjach zmieniające wartość licznika.
4. Wyjaśnij zaobserowane wyniki widoczne w konsoli
5. BONUS: Napisz test z wykorzystaniem biblioteki JUnit sprawdzający że Counter jest poprawnie stworzonym singletonem. Wykorzystaj asercję assertSame

Autor: Michał Bojanowski

Ćwiczenie 2 (pl.sdacademy.designpatterns.signleton.lazy)

1. Stwórz singleton (type EAGER) AppConnections która posiada 3 pola: connectedUsers(List<String>), timeout(long) i currentConnectionsNum(int = 0)
2. Dodaj do stworzonego singletonu metodę connectUser(String username), która zwiększa ilość currentConnectionsNum i dodaje użytkownika do listy connectedUsers.
3. Dodaj gettery dla wszystkich pól
4. W metodzie main, stwórz dwie referencje klasy AppConnections. Dodaj po jednym użytkowniku wykorzystując każdą z nich. Porównaj konfiguracje z obu referencji i wyjaśnij to zachowanie.
5. Zmień implementację singletonu z EAGER na LAZY
6. BONUS: Napisz testy z wykorzystanie JUnit potwierdzające, że AppConfiguration jest singletonem i wywołania metody getConnectedUsers na różnych referencje klasy AppConfiguration zwracają tę samą listę.

Autor: Michał Bojanowski

Ćwiczenie 3 (pl.sdacademy.designpatterns.signleton.enumm)

1. Stwórz singleton (typu ENUM) UsersRepository który posiada 1 pole: usernamesToEmails (typu Map<String, String>). Dodaj getter dla tego pola.
2. Dodaj do stworzonego singletonu metodę addUser(String username, String email), która dodaje do mapy użytkownika jako klucz i jego email jako wartość
3. Dodaj interfejs funkcjonalny RegistrationService z metodą register(String username, String email)
4. Dodaj klasy FBRegistrationService i GoogleRegistrationService, które implementują interfejs RegistrationService i wykorzystują UsersRepository do przechwywania użytkowników.
5. W metodzie main Stwórz instancję FBRegistrationService i GoogleResitrationService. Za ich pomocą zarejestruj kilku użytkoników i innej nazwie. Na koniec stwórz referencję singletonu UsersRepository, wyświetl zawartość mapy usernamesToEmails i wyjaśnij rezultat.

Autor: Michał Bojanowski

Słów kilka singletonie dla wielu wątków

```
public class ThreadSafeSingleton {

    private static ThreadSafeSingleton instance;

    private ThreadSafeSingleton() {
    }

    public static ThreadSafeSingleton getInstance() {
        if (instance == null) {
            synchronized (ThreadSafeSingleton.class) {
                if (instance == null) {
                    instance = new ThreadSafeSingleton();
                }

            }
        }
        return instance;
    }
}
```

Autor: Michał Bojanowski

Singleton - Pytania

1. Do czego służy singleton?
2. Jakie są wady korzystania z singletonu?
3. Jakie są sposoby korzystania z singletonu?
4. Które z tych sposobów możemy używać w aplikacjach wielowątkowych?

Autor: Michał Bojanowski

Builder

Problem?
Jak stworzyć obiekt który posiada wiele pól?
Jak stworzyć taki obiekt kiedy chcemy zainicjalizować tylko część z nich?

Rozwiązanie

- ~~napisać dla każdej wersji osobny konstruktor~~
- ~~stworzyć konstruktor dla wszystkich pól i przekazywać nulle~~
- wykorzystać Builder Pattern czyli stworzyć klasę, która daje dostęp do ustawiania wartości dowolnych pól ’po kropce’

Autor: Michał Bojanowski

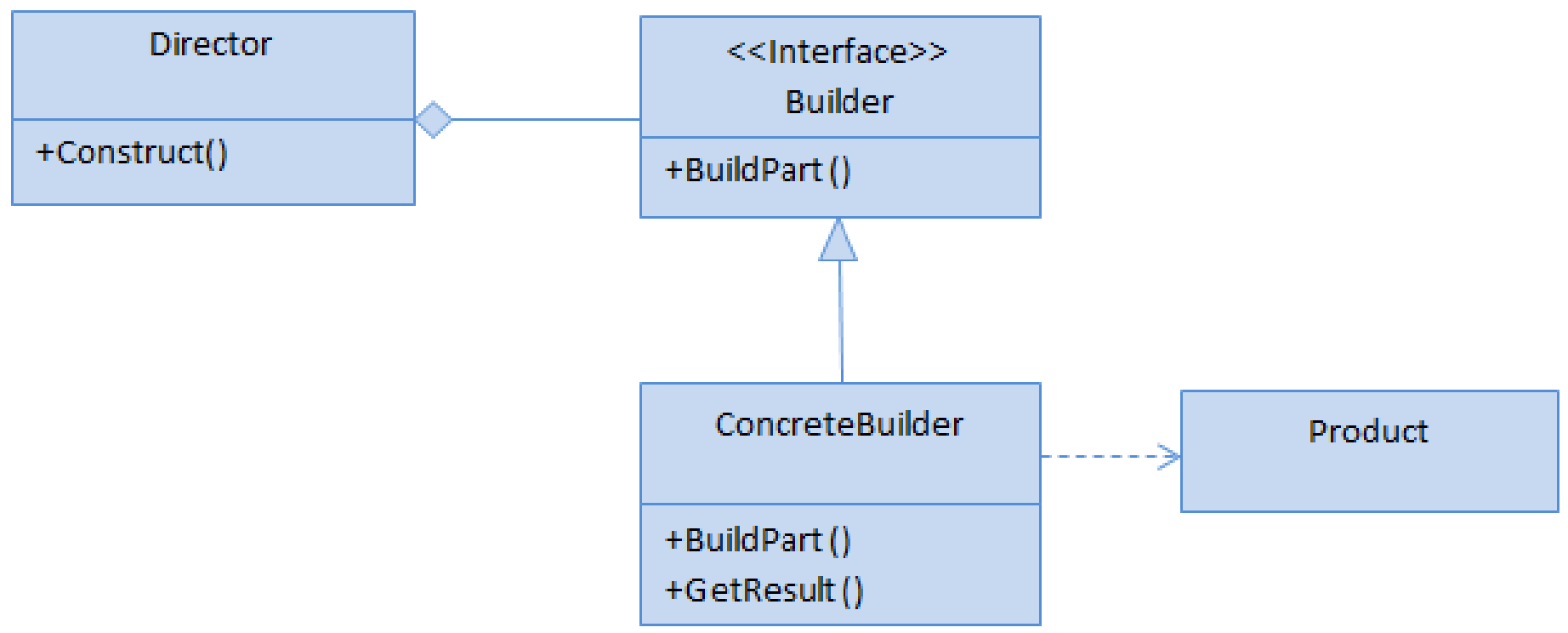
Builder

Jak uzyskać dostęp do **aktualnie** używanego obiektu aby po wywołaniu **jednej** metody, móc wywołać na tym samym obiekcie wywołać **kolejną** metodę?

this

Autor: Michał Bojanowski

Builder



Builder Pattern

Ćwiczenie 4 (pl.sdacademy.designpatterns.builder)

1. Stwórz klasę Player składającą się z 5 pól: health (int), mana(int), nick(String), level(long) friends(List<String>)
2. Stwórz klasę statyczną Builder wewnątrz klasy Player, która ma 5 pól jak klasa Player
3. Stwórz 6 metod:
   - 5 metod ustawiających ustawiających konkretne pole i zwracające ’siebie’
   - metodę ’build’ która tworzy ostateczny obiekt wykorzystując ustawione wartości w klasie Builder
4. W metodzie main wykorzystaj stworzony Builder do stworzenia kilku różnych obiektów player.

Autor: Michał Bojanowski

Ćwiczenie 5 (pl.sdacademy.designpatterns.builder)

1. Stwórz klasę Book składającą się z 5 pól: ISBN(String), pages(int), title(String), author(String) description(String)
2. Stwórz klasę BookBuilder , która ma 5 pól jak klasa Player
3. Stwórz 7 metod:
   - 5 metod ustawiających ustawiających konkretne pole i zwracające ’siebie’
   - metodę ’build’ która tworzy obiekt Book wykorzystując ustawione wartości w klasie BookBuilder
   - motodę toString w klasie Book
4. W metodzie main wykorzystaj stworzony BookBuilder do stworzenia kilku różnych książek. Wypisz dane każdej książki na ekranie.

Autor: Michał Bojanowski

Ćwiczenie 6 (pl.sdacademy.designpatterns.builder)

1. Stwórz klasę Person składającą się z 3 pól: name(String), surname(String), age(int)
2. Oznacz klasę Person adnotacją @Builder z biblioteki lombok
3. W metodzie main stwórz obiekt Person z wykorzystaniem statycznej metody builder() na obiekcie Person.
4. Dodaj odpowiednie pola w adnotacje @Builder aby zmienić nazwy method tworzących builder i tworzących zbudowany obiekt

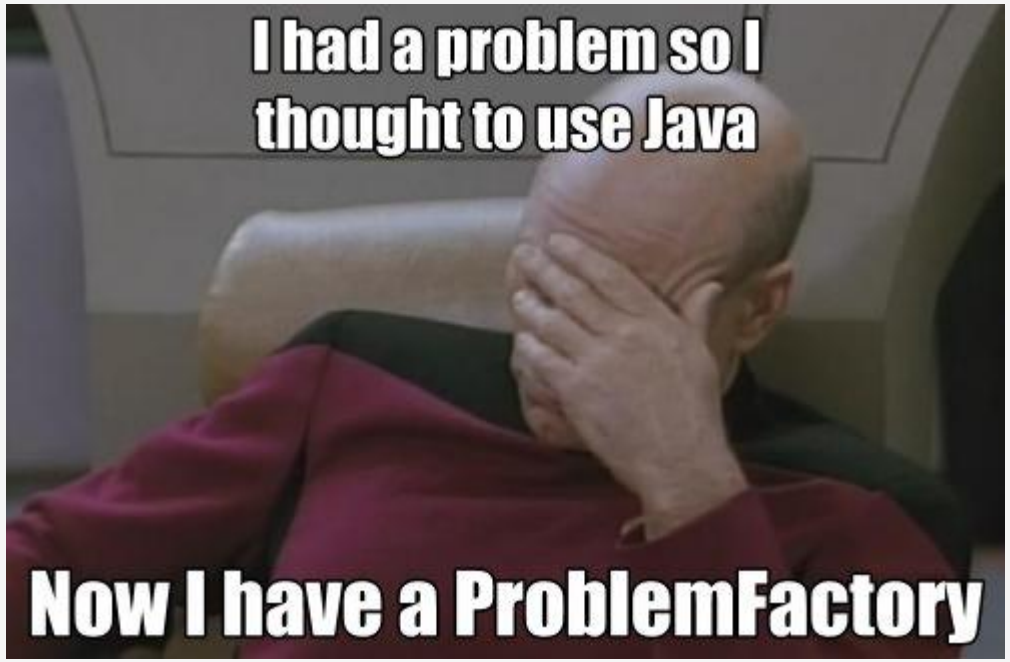
Autor: Michał Bojanowski

Builder - pytania

1. Do czego służy wzorzec Builder?
2. Kiedy warto rozważyć użycie wzorca Builder?
3. W jakim przypadku lepiej użyć konstruktora niż wzorca Builder?

Autor: Michał Bojanowski

Abstract Factory design pattern



Autor: Michał Bojanowski

Abstract Factory design pattern

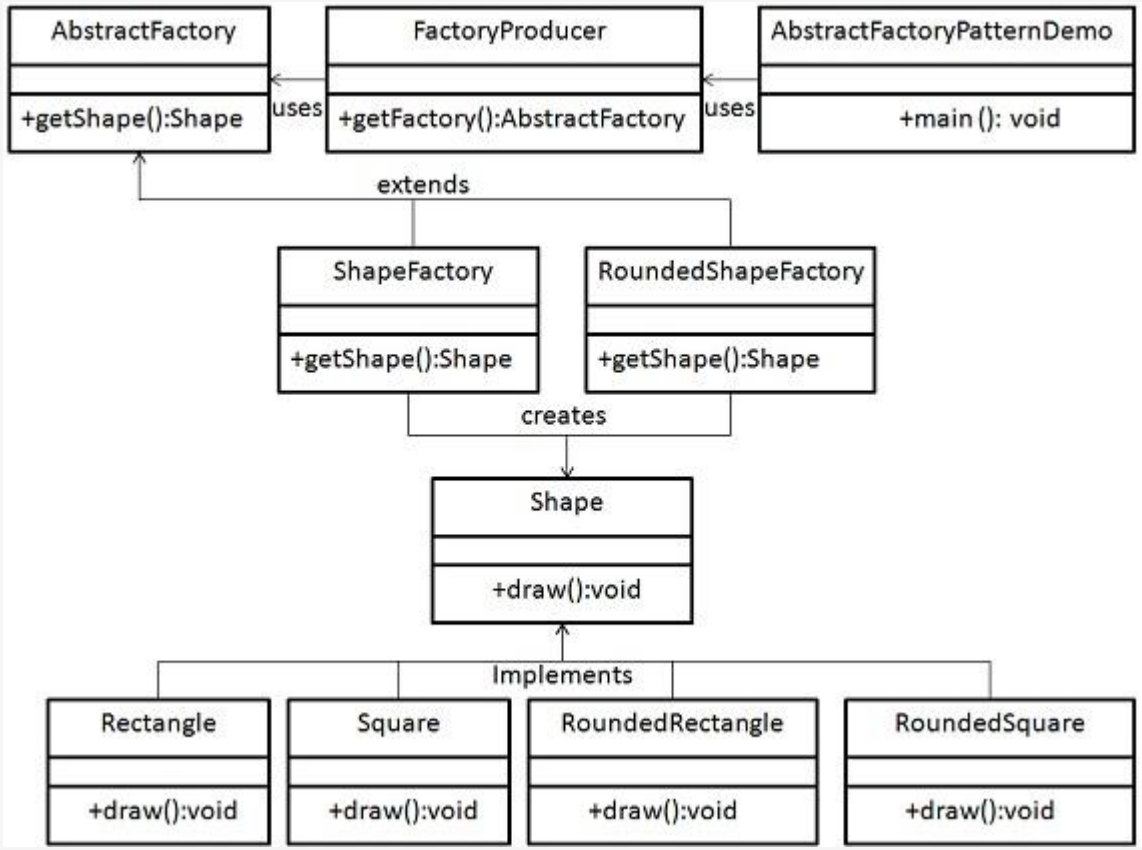
Problem?
Jak wielokrotnie stworzyć (najczęściej skomplikowany) obiekt (czyli taką klasę która ma wiele pól), który ma ustalone wartości poszczególnych pól/konkretną implementację metod?

Stworzyć **fabrykę** która wyprodukuje nam konkretny obiekt za pomocą jednego prostego wywołania.

Przykład?
’Wyprodukuj pizzę farmerską’ (ciasto, ser, boczek, kiełbasa, cebula, papryka...)
’Wyprodukuj VW Golf’ (4 koła, klimatyzacja automatyczna, silnik 1.5 TSI, radio,...)

Autor: Michał Bojanowski

Abstract Factory design pattern



Abstract Factory design pattern

Kiedy używać AbstractFactory?

- gdy klient chce być niezależny od tego jak tworzymy obiekt
- gdy system zawiera wiele rodzin obiektów które są używane najczęściej razem
- gdy podczas **runtime** musimy stworzyć konkretną zależność

Autor: Michał Bojanowski

Ćwiczenia 7 (pl.sdacademy.designpatterns.abstractfactory.factory)

Stwórz interfejs AbstractFactory<T> a w nim T create(String type)
Stwórz paczkę car a w niej:

- enum EngineType z wartościami GASOLINE_NATURALLY_ASPIRATED, GASOLINE_TURBO, DIESEL
- enum BodyType z wartościami SEDAN, HATCHBACK, COMBI
- interfejs Car z 5 metodami

```
String getModelName();
  EngineType getType();
  Double getEngineVolume();
  int getCylindersNum();
  BodyType getBodyType();
```

- dwie **niepubliczne** klasy implementujące ten interfejs
- klasę SedanCarFactory implementującą interfejs AbstractFactory. Wewnątrz metody create wykonaj switch na typie i na podstawie tego stwórz konkretny samochód typu sedan
- W przypadku nieznanego kodu wyrzuć dowolny wyjątek z odpowiednią wiadomością

Autor: Michał Bojanowski

Ćwiczenia 7 CD

- stwórz klasę CombiCarFactory implementującą interfejs AbstractFactory. Wewnątrz metody create wykonaj switch na typie i na podstawie tego stwórz konkretny samochód typu combi

- Stwórz w bazowej paczce projekty abstractfactory enum FactoryCategory z wartościami SEDANS, COMBIS
- Stwórz klasę FactoryProvider która ma statyczną metodę AbstractFactory getFactory(FactoryCategory category) w której na podstawie kategorii stworzysz odpowiednią implementację factory. Przetestuj klasę FactoryProvider w metodzie main

Autor: Michał Bojanowski

Ćwiczenia 8 (pl.sdacademy.designpatterns.abstractfactory.factory)

- stwórz klasę HatchbackCarFactory implementującą interfejs AbstractFactory. Wewnątrz metody create wykonaj switch na typie i na podstawie tego stwórz konkretny samochód typu hatchback
- dodaj potrzebny kod aby stworzyć samochód typu hatchback w metodzie main (tzn. dodaj wartość do FactoryCategory i możliwość wybory fabryki w klasie FactoryProvider

Autor: Michał Bojanowski

Pytania

1. Jakie wzorce projektowe mówią nam jak tworzyć obiekty?
2. Którzy wzorzec wykorzystasz do stworzenia obiektu który ma jedną instancję w całej aplikacji?
3. Jakie są sposoby tworzenia singletonu?
4. Jaki wzorzec/wzorce projektowe wykorzystasz do stworzenia obiektu, który składa się z wielu pól
5. Do czego służy wzorzec abstract factory design pattern?

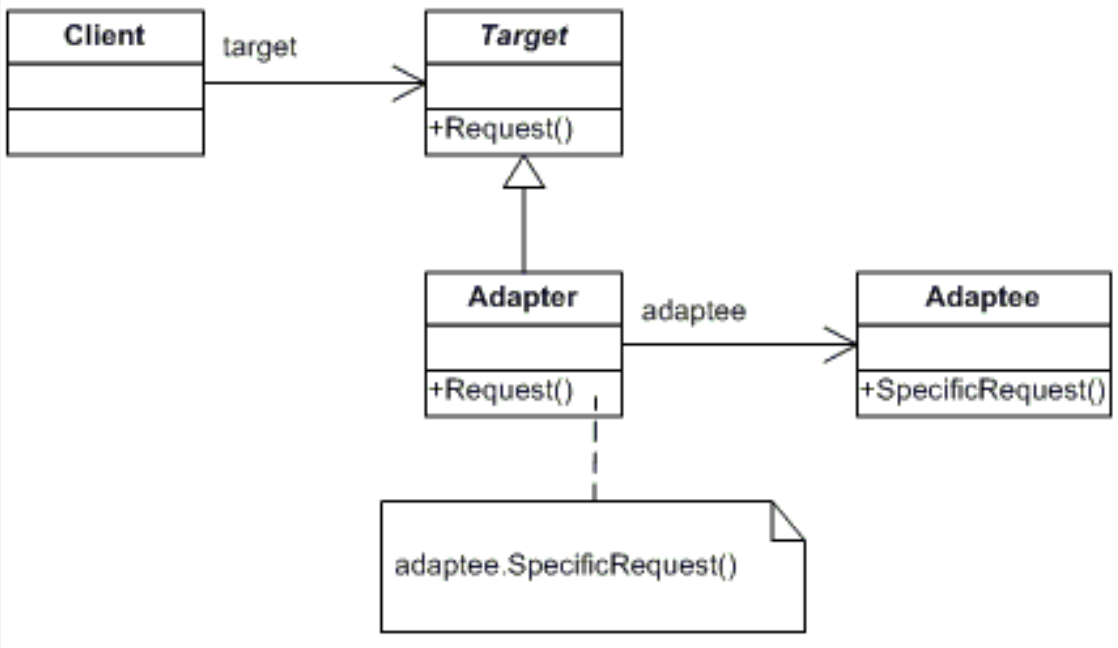
Autor: Michał Bojanowski

Adapter

Problem?
Integrujemy np. duży system, który pobiera dane użytkowników z wielu zewnętrznych systemów jednakże każda implementacja przedstawia te dane w zupełnie inny sposób i implementuje inny interfejs.

Rozwiązanie?
Zastosować Adapter, czyli warstwę pośrednią, która dostosowuje dany interfejs do docelowego

Adapter



Autor: Michał Bojanowski

Ćwiczenia 9 (pl.sdacademy.designpatterns.adapter)

- stwórz interfejs User z 4 metodami

```
String getFullname();
String getUsername();
Integer getAge();
List<String> getRoles();
```

- stwórz paczkę systema a w niej klasę SystemAUser z polami

```
private String firstName;
private String lastName;
private long age;
private List<String> roles;
```

- stwórz paczkę systemb a w niej klasę SystemBUser z polami

```
private String fullName;
private String displayName;
private String age;
private Set<String> roles;
```

- Dodaj do obu klas getter, settery, konstruktor i metodę toString

Autor: Michał Bojanowski

Ćwiczenia 9 - CD

- stwórz adapter klasy SystemAUser do interfejsu User
- stwórz adapter klasy SystemBUser do interfejsu User

- BONUS: stwórz klasę SystemCUser, która ma pola

```
private String firstName;
private String lastName;
private long birthDateTimeStamp;
private String role;
```

a następnie adapter klasy SystemCUser do interfejsu User

Autor: Michał Bojanowski

Ćwiczenia 10 (pl.sdacademy.designpatterns.adapter)

- stwórz paczkę systemd a w niej klasę SystemDUser z polami
  - birthDate(LocalDate)
  - role(String)
  - alias(String)
  - name(String)
- Stwórz adapter klasy SystemDUser do interfejsu User

Autor: Michał Bojanowski

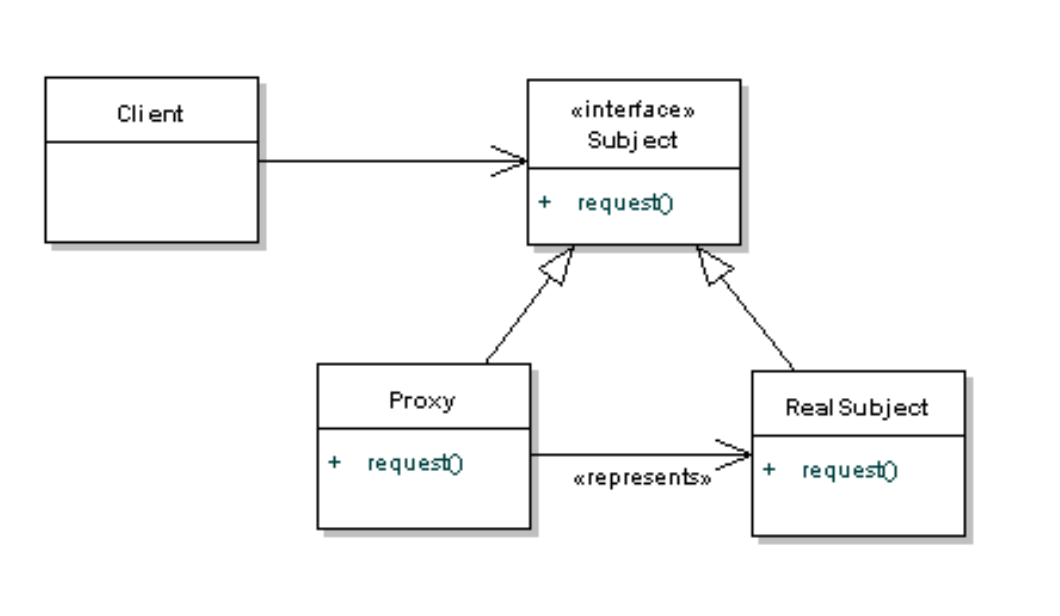
Proxy

Problem?
Chcemy stworzyć obiekt ale ukryć fragmenty implementacji. (Virtual Proxy)
LUB
Chcemy uprościć potencjalnie skomplikowany obiekt
LUB
Chcemy ukryć tworzenie ciężkiego obiektu przed użytkownikiem
LUB
Chcemy trzymać reprezentację obiektu będącego w innej przestrzeni - w naszej lokalnej (Remote Proxy)
LUB
Chcemy dodać dodatkową warstwę np. bezpieczeństwa na konkretny obiekt (Security Proxy)

Autor: Michał Bojanowski

Proxy

Rozwiązanie? Proxy, czyli stworzenie warstwy pomiędzy interfejsem a prawdziwym obiektem implementującym ten interfejs, która TEŻ implementuje dany interfejs.



Autor: Michał Bojanowski

Ćwiczenia 11 (pl.sdacademy.designpatterns.proxy)

CEL ZADANIA

Stwórzmy proxy która odpowiada za tworzenie ’ciężkiej’ konfiguracji i ukrywa ten fakt przed użytkownikiem.

Autor: Michał Bojanowski

Ćwiczenia 11 (pl.sdacademy.designpatterns.proxy)

- stwórz interfejs ConfigLoader z metodą String load();
- stwórz implementację klasy ConfigLoader. Niech klasa ta ma jedno pole configuration(String) i w jednoargumentowym konstruktorze, w którym podajemy serverUrl, sumulujemy pobieranie konfiguracji z tego serwera (niech konstruktor śpi przez 2 sekundy i wypisuję informację o pobraniu konfiguracji na końcu konstruktora). metoda load powinna zwracać konfigurację zapisaną w konstruktorze.
- stwórz proxy implementującą interfejs ConfigLoader. W metodzie load zwracaj konfigurację zwróconą przez klasę stworzoną w poprzednim punkcie tylko jeżeli nie została ona pobrana z zewnętrznego serwera (tzn. z serverUrl)
- w metodzie main zawołaj metodę load na stworzonym obiekcie proxy więcej niż raz. Co możesz powiedzieć o czasu wykonania poszczególnych wywołań metody call?

Autor: Michał Bojanowski

Ćwiczenia 12

CEL ZADANIA

Stwórzmy proxy która ogranicza prawa do używania danego interfejsu w zależności od uprawnień użytkownika

Autor: Michał Bojanowski

Ćwiczenia 12 (pl.sdacademy.designpatterns.proxy.security)

- stwórz paczkę security i dodaj enum AccessType z wartościami READ i WRITE
- stwórz enum Role z wartościami USER i ADMIN. Każda Rola posiada listę AccessType. USER podsiada READ, ADMIN posiada READ i WRITE
- Zdefiniuj klasę Person która ma pola
- stwórz klasę BuilderPerson

(lub użyj @Builder)

```
private String name;
private String email;
private String phone;
private LocalDate birthDate;
private Role role;
private int numOfLogins;
private boolean verified;
```

- stwórz interfejs PersonManager:

```
void addPerson(Person person);
void deletePerson(String emaial);
List<Person> getAllPeople();
boolean isPersonPresent(String name);
void validateEmail(String email);
```

Autor: Michał Bojanowski

Ćwiczenia 12 CD

- stwórz implementację interfejsu PersonManager. addPerson powinien wyrzucić wyjatek w przypadki zduplikowanego maila w wewnętrzej liście elementów typu Person
- stwórz Proxy typu Security dla implementacji stworzonej punkt wyżej, która przechowuje instancję klasy Person - osobę która zarządza listą Person
- wywołaj przez proxy metody zmieniające zawartość listy użytkowników jeżeli zarządzający ma WRITE access
- wywołaj przez proxy metody proszące o zawartość jeżeli zarządzający ma READ access

Autor: Michał Bojanowski

Decorator

Problem?
Podczas działania programu, w zależności od pewnych zmiennych (np. środowisko) chcielibyśmy bazowy obiekt rozszerzać o pewne funkcjonalności

Rozwiązanie?
Zastosować decorator, czyli obiekt który rozszerza funkcjonalność klasy bazowej poprzez rozszerzenie klasy bazowej

Autor: Michał Bojanowski

Decorator w javie

```
new LineNumberReader(new BufferedReader(new FileReader(new File(""))));
new FileReader().read();
new BufferedReader().read();
new LineNumberReader().read();
```

BufferedReader dekoruje obiekt FileReader
LineNumberReader dekoruje BufferedReader

Autor: Michał Bojanowski

Decorator - type

- Dodające funkcjonalność (nowe metody)
- Rozszerzające metody klasy bazowej/interfejsu

Jakim typem dekoratora jest BufferedReader dla klasy FileReader i dlaczego?

Autor: Michał Bojanowski

Decorator

Plusy?

- jasne zastosowanie
- nietrudna implementacja

Minusy?

- możliwa spora ilość klas dekorujących
- możliwe zmniejszona czytelność przy zastosowaniu wielu dekoratorów

Autor: Michał Bojanowski

Ćwiczenia 13 (pl.sdacademy.designpatterns.decorator)

- stwórz interfejs EngineStarter z metodami:

```
void startEngine();
void stopEngine();
```

- stwórz implementację tego interfejsu - Car, która odpowiednio zmienia wartość pola typu boolean
- stwórz dekorator CarEngineStatusBeforeLoggingDecorator, który przed włączeniem i wyłączeniem silnika loguje na ekran odpowiednią informację
- stwórz dekorator CarEngineStatusAfterLoggingDecorator, który po włączeniu i wyłączeniu silnika loguje na ekran odpowiednią informację
- w metodzie main wykorzystaj oba stworzone dekoratory aby na obiekcie Car po wywołaniu metody startEngine została zalogowana na ekranie informacja przed i po włączeniu silnika

Autor: Michał Bojanowski

Ćwiczenia 14 (pl.sdacademy.designpatterns.decorator)

- Stwórz klasę abstrakcyjną PizzaBase i metodą displayPizzaContent która zwraca List<String> i dla tej klasy zwraca jeden element ’Base’
- Stwórz dwa dekoratory klasy PizzaBase - ThickPizzaBase i FlatPizzaBase które modyfikują wartość ’Base’ w zawartości klasy PizzaBase o odpowiedni przedrostek
- Stwórz dekorator klasy PizzaBase - WithCheesePizza. Dodaj ser do składników.
- Stwórz dekorator klasy PizzaBase - WithMushroomPizza. Dodaj pieczarki do składników
- Stwórz dekorator klasy PizzaBase - WithSalamiPizza. Dodaj salami do składników
- Stwórz dekorator klasy PizzaBase - WithHamPizza. Dodaj cebulę i szynkę do składników.

- W metodzie main stwórz pizzę z szynką i pizzę salami za pomocą stworzonych dekoratorów. Wyświetl składniki stworzonej pizzy.

Autor: Michał Bojanowski

STOP! Przypomnienie

1. Jaki wzorzec użyć do dostosowania niepasującej implementacji do konkretnego interfejsu?
2. Jaki wzorzec projektowy użyć do dołożenia warstwy pomiędzy obiektem a jego klientem, który np. ogranicza możliwość wykonania pewnych operacji w zależności od praw użytkownika?
3. Jaki wzorzec projektowy użyć do zalogowania wejścia do każdej metody publicznej w danej klasie?
4. Jaki wzorzec projektowy użyć do stworzenia pojedynczej instancji klasy w całej aplikacji?
5. Jaki wzorzec projektowy użyć do stworzenia skomplikowanego obiektu, w którym chcę ustawić wartości tylko części pól tej klasy?

Autor: Michał Bojanowski

Facade

Problem?
Jak uprościc ciąg wywołań potrzebny do uzyskania pewnego efektu?

Rozwiązanie
Stworzyć **fasadę** czyli klasę która upraszcza interfejs i upraszcza wykonanie pewnej operacji.

Przykład:

- zadanie administratora: przygotuj backupy
- dostępne operacje administratora: skopiuj backup na serwerA, skopiuj plik na serwerB, skopiuj plik na serwerC
- Fasada: skopiuj backupy na serwery (na serwer A,B,C)

Autor: Michał Bojanowski

Facade

Głowne zadania

- UPROSZCZENIE INTERFEJSU
- SCHOWANIE POZIOMU KOMPLIKACJI ZA PEWNĄ WARSTWĄ

Autor: Michał Bojanowski

Ćwiczenia 15 (pl.sdacademy.designpatterns.facade)

- Stwórz klasę Product z polami: name (String), id(String), price(String).
- Stwórz klasę AvailabilityService z metodą boolean isAvailable(Product product), która sprawdza czy produkt jest dostępny (niech zwrócony boolean będzie losowy)
- Stwórz klasę PaymentService z metodą pay(List<Product> product która przyjmuje listę produktów i wypisuje sumę zakupów na ekran i następnie ją zwraca
- Stwórz klasę ProductSearchService która zawiera statyczną listę produktów (stwórz 5 dowolnych typów produktów) i metodę searchById która zwraca Optional<Product>
- Stwórz Fasadę do robienia zakupów z metodą buy, która przyjmuje listę id produktów które użytkownik chce kupić. Fasada powinna dokonać zakupu dostępnych produktów i zwracać sumę do zapłacenia.
- Użyj stworzoną fasadę w metodzie main do zrobienia zakupów. Lista id produktów powinna przyjść w liscie argumentów programu.

Autor: Michał Bojanowski

Observer

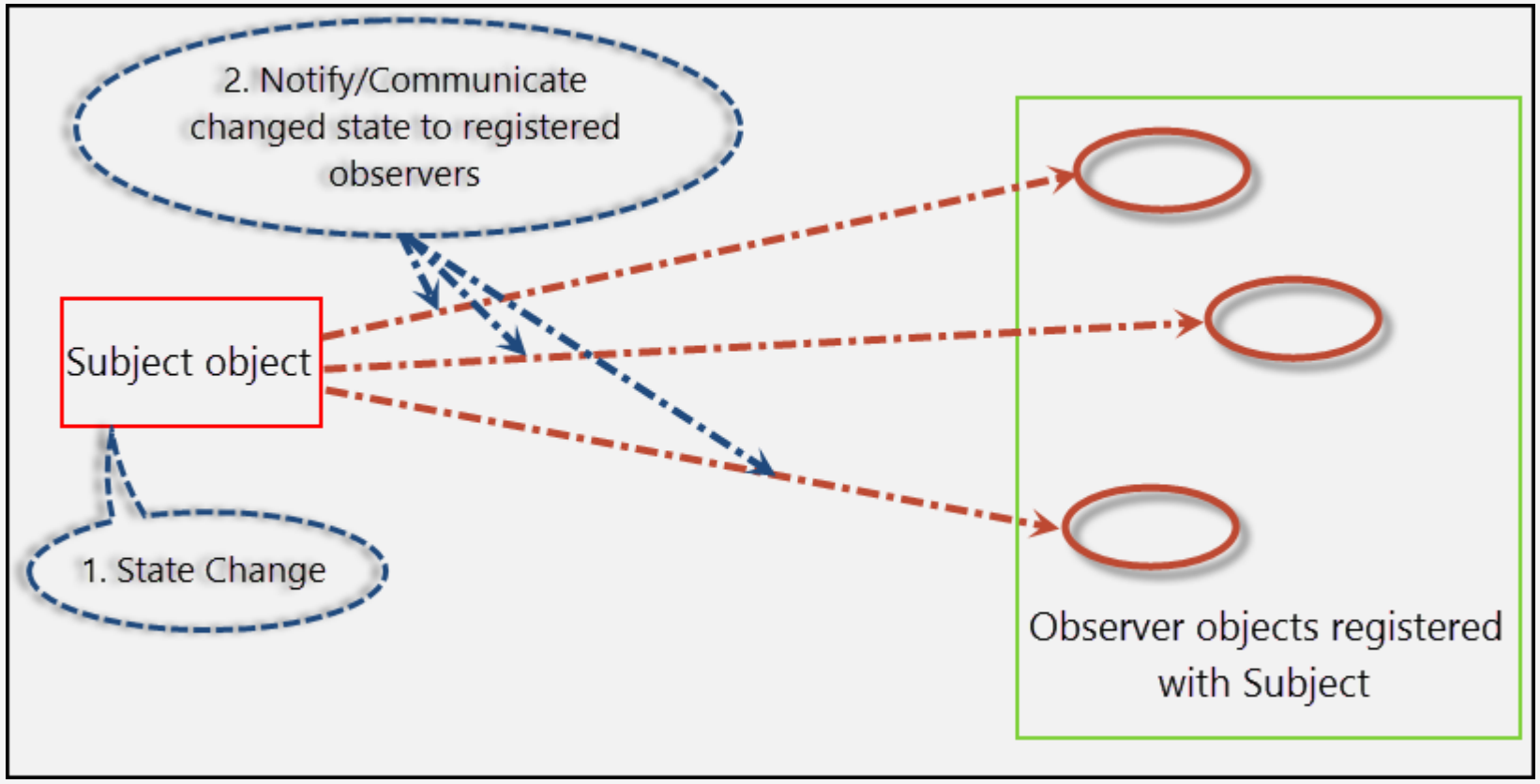
Problem?
Jak stworzyć relację jeden do wielu zależną w taki sposób, że aktualizacja pojedycznego obiektu aktualizuje wszystkie zależne jednocześnie

Rozwiązanie?
Zastosować Observer Pattern, tzn. połączyć zależne obiekty (Observery) z danym tematem (Subject) w taki sposób, że Subject ma wiedzę nt. wszystkich obserwujących (dzięki czemu może ich poinformować o zmianach stanu)

Przykład:
- Kanał na slacku!
- JMS

Autor: Michał Bojanowski

Observer



Autor: Michał Bojanowski

Ćwiczenia 16 (pl.sdacademy.designpatterns.observer)

- Stwórz klasę Subject, która ma pola: List<Observer> i value(int), i ma możliwość zarejestrowania Observera, zaktualizowanie pola value i poinformowaniu wszystkich obserwerów o aktualizacji wartości
- Stwórz klasę abstrakcyjną Observer, która ma pole *subject(Subject)* i abstrakcyjną metodę *void update()*
- Stwórz 3 implementacje klasy Observer, które w konstruktorze przyjmują Subject i subskrybują się do niego
- Stwórz klasę ConcreteValueObserver, która w metodzie update() zawsze wypisuje wartość na ekran
- Stwórz klasę ValueLoweredObserver która w metodzie update() wypisuje wartość na ekran tylko gdy zaktualizowana wartość jest mniejsza niż poprzednia
- Stwórz klasę ByTenChangedObserver, która w metodzie update() wypisuje wartość na ekran jeżeli nowa wartość różni się przynajmniej o 10 od poprzedniej
- W metodzie main stwórz Subject, każdy z typów observerów. Zasubsrykój je do obiektu typu Subject. Następnie kilka razy zaktualizuj wartość na obiekcie Subject i poinformuj wszystkich Obserwów o zmianach

Autor: Michał Bojanowski

Ćwiczenia 17 (pl.sdacademy.designpatterns.observer)

- Wykorzystując wzorzec Observer stwórz implementację sumulującą rozmowę wielu użytkowników na Slacku
- Stwórz klasę Chat (jako Subject) zawierający listę wiadomości na kanale
- Stwórz klasę User (jako Observer), który może dołączyć do czatu i obserwować kolejne wiadomości na kanale, a także wysłać wiadomość na czat
- W metodzie main stwórz czat, kilku użytkowników i przetestuj swoją implementację

- Zmień implementację w taki sposób aby wykorzystać Obiektu typu Observer i Observable z paczki java.util

Autor: Michał Bojanowski

Command

Problem?
Jak stworzyć obiekt na którym można wywołać dany efekt ale również go **cofnąć** w dowolnym momencie?

Rozwiązanie
Zastosować komendę, tzn. opakować dany proces w obiekt który może np. wykonać proces lub go cofnąć do pierwotnego stanu.

Autor: Michał Bojanowski

Command

Przykłady:

- skopiuj pliki z katalogu A do katalogu B (tzn. wykonaj proces)
- usuń pliki z katalogu B przed chwilą skopiowane z katalogu A (tzn. config efekt wykonania procesu)

- wystartuj film (tzn. wykonaj proces)
- zastopuj film (tzn. config efekt wykonanego procesu)

- podczas kupowania produktów w sklepie, po podaniu danych przejdź do etapu wyboru metody płatności (tzn. wykonaj proces)
- wróć do etapu podawania danych (tzn. config efekt wykonanego procesu polegającego na potwierdzeniu danych)

- git i opcja revert lub reset --soft

Autor: Michał Bojanowski

Ćwiczenia 18 (pl.sdacademy.designpatterns.command)

- Stwórz interfejs CommandBase z metodami execute() i undo()
- Stwórz POJO UserData z polami name, surname, email, password (wszystkie typu String)
- Stwórz klasę UserRepository, która przechowuje listę użytkowników. Dodaj metodę addUser(UserData user) i removeUser(UserData userData). Dodaj metodę toString która wyświetla przynajmniej imiona wszystkich użytkowników.
- stwórz klasę UserRegistrationCommand która implementuje interfejt CommandBase. W konstruktorze tej klasy wykorzystaj UserRepository i UserData
- w metodzie main stwórz instancję klasy UserRepository, dwóch użytkowników i komendę dla każdego z nich, wykonaj obie komendy, wyświetl użytkowników w repozytorium, a następnie cofnij ich wykonanie i potwierdź fakt braku użytkowników w repozytorium

Autor: Michał Bojanowski

Ćwiczenia 19 (pl.sdacademy.designpatterns.command)

Wykorzystując wzorzec projektory Command, stwórz klasę odzwiercierdlającą zachowanie pilota do telewizora

- dodaj możliwość zmiany kanału,
- dodaj możliwość powrotu na poprzedni nr kanału.

Przykład oczekiwanego zachowania:

- Pierwszy włączony kanał to nr 5. Powrót na poprzedni kanał nie zmienia aktualnego kanału
- Kanał zostaje przełączony na nr 8. Powrót na poprzedni kanał zmienia aktulny kanał na nr 5. Wykonanie akcji powrót po raz kolejny powoduje przełączenie kanału na nr 8.
- Przetestuj to zachowanie w metodzie main()
- BONUS: przetestuj to zachowanie za pomocą testów napisanych z wykorzystaniem biblioteki JUnit.

Autor: Michał Bojanowski

Memento

Problem?
Jak zachować stan obiektu (bez naruszenia zasad SOLID) w taki sposób aby móc w dowolnym momencie przywrócić stan obiektu.

Przykładowe problemy rozwiązywane przez design pattern Memento
- zapisywanie stanu gry
- zapisywanie stanu pliku w edytorze tekstkowym

Ważne!
- klasa implementująca stan obiektu który przechowuje powinna mieć takie same pola

Autor: Michał Bojanowski

Ćwiczenia 20 (pl.sdacademy.designpatterns.memento)

- Stwórz klasę (EditorText) która przechowuje aktualną zawartość pola tekstowego w edytorze tekstowym. Klasa ta posiada jedno pole z aktualna wartością tekstu na ekranie(String). Dodaj metodę addText, która dodaj wartość do zapisanej wartości tesktu
- Stwórz klasę EditorTextMemento bedąca obiektem Memento dla klasy EditorText
- Stwórz klasę EditorTextMementoManager która zarządza listą memento jako stos. Niech memento będzie typu ArrayDeque. Dodaj metodę save i restore (która odpowiednio zmienia ArrayDeque wewnątrz klasy)
- W klasie EditorText dodaj metodę restoreFromMemento, która przywraca stan wartośc tekstu edytora
- w metodzie main przetestuj swoją implementację zmieniając i zapisując wartość edytora tekstowego kilka razy, a następnie przywracając kolejne zapisane stany.

Autor: Michał Bojanowski

Ćwiczenia 21 (pl.sdacademy.designpatterns.memento)

- Zmień sposób przechwywania danych w klasie EditorText z poprzedniego ćwiczenia. Niech String będzie zastąpiony przez mapę, która danemu nr linii (zaczynając od 1) będzie przypisywała tekst występujący w linii o danym numerze. Dostosuj resztę klas w tej paczce do tej zmiany

Autor: Michał Bojanowski

Strategy

Problem?
Jak zdefiniować grupę algorytmów które można zamiennie wykorzystać w czasie wykonywania się programu

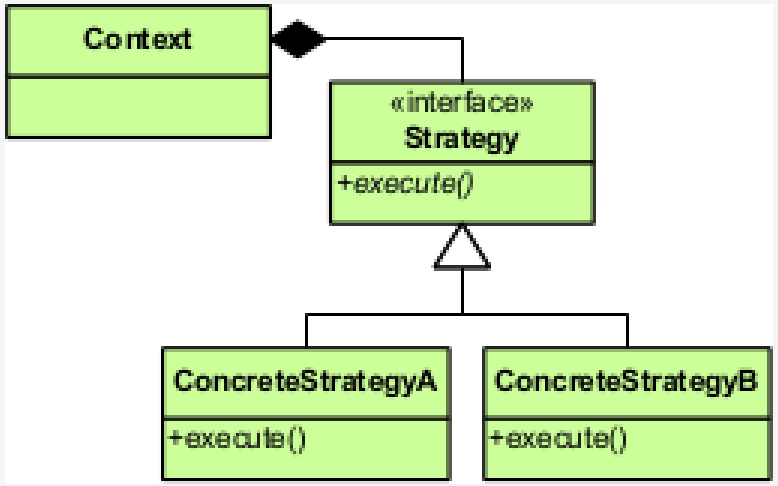
Rozwiązanie
Wykorzystać strategię, tzn. w zależności od czynników zewnętrznych (konfiguracja, wybór użytkownika) wybrać daną implementację algorytmu

Przykłady:
- w zależności od systemu wybierz inny domyślny rozszerzenie pliku z certyfikatami
- w zależności od dostępnego czasu wybierz inną siłe algorytmu hashującego hasło
- w zależności od trybu systemu (recovery, safe mode, normal) wybierz wygląd okienek

Autor: Michał Bojanowski

Strategy

- W zależności od kontekstu, klient wybiera odpowiednią strategię



Autor: Michał Bojanowski

Ćwiczenie 22 (pl.sdacademy.designpatterns.strategy)

- Stwórz interfejs SecretKeyGenerationStrategy, który posiada metodę generate() zwracającą obiekt SecretKey
- Stwórz 3 strategie generowania SecretKey: AES, DES i HmacSHA256. Każda implementacja w metodzie generate() tworzy KeyGenerator odpowiedniego typu, inicjuje go obiektem SecureRandom i następnie zwraca wygenerowany klucz
- Stwórz klasę SecretKeyGenerator, która w konstruktorze przyjmuje instancję obiektu implementującego interfejs SecretKeyGenerationStrategy, i posiada metodę create(), która zwraca wygenerowany SecretKey odpowiednią strategią.
- Dodaj metodę do klasy SecretKeyGenerator pozwalającą zmieniać strategię generowania SecretKey.
- W metodzie main przetestuj generowanie kluczy za pomocą klasy SecretKeyGenerator i różnych strategii, wypisz na ekran algorytm jaki został użyty do stworzenia danego klucza

Autor: Michał Bojanowski

TEMPLATE PATTERN

PROBLEM?

- Jak stworzyć szkielet algorytmu w taki sposób aby łatwo podmieniać jego **CZĘŚCI?**

Rozwiązanie
Zaimplementować szkielet algorytmu (najczęściej w klasie abstrakcyjnej), podzielić go na części (najczęściej na osobne metody) w taki sposób aby w klasach dziedziczących nadpisać interesujące dla nas części

Przykład
Algorytm opisujący działanie linii produkcyjnej wytwarzające dwa modele samochodów na tym samym typie podłogi

Autor: Michał Bojanowski

Ćwiczenia 23 (pl.sdacademy.designpatterns.template)

Stwórz szkielet testu wydajnościowego

- stwórz klasę abstrakcyjną PerformanceTestTemplate a niej 3 metody abstrakcyjne
  - void testIteration() - wykonująca pojedynczą iterację testu
  - int getWarmupIterationsNum() - zwracająca ilość iteracji rozgrzewających JVM
  - int getIterationsNum() - zwracająca ilość iteracji w teście
- dodaj metodę void run() która najpierw wykonuje rozgrzewkowe iteracje testu, a następnie właściwe iteracje testu
- zmierz i wyświetl średni czas wykonania pojedynczej iteracji

Autor: Michał Bojanowski

CD Ćwiczenie 23

Dodaj implementację klasy PerformanceTestTemplate która w czasie pojedynczej iteracji:

- generuje listę 10000 losowych liczb
- sortuje wartości w liscie, od najmniejsze do największej

Dodaj kolejną implementację klasy PerformanceTestTemplate która w czasie pojedynczej iteracji:

- generuje string za pomocą konkatenacji w taki sposób że 10000 dodaje w niego losową cyfrę

W metodzie main uruchom oba testy.

Autor: Michał Bojanowski

INTERPRETER

Problem?
Jak stworzyć obiekt, który nas zrozumie? ☺
Jak przetłumaczyć nasze ’komendy’ na konkretne polecenia dla programu?

Rozwiązanie
Stworzyć interpreter czyli obiekt który potrafi przetworzyć nasze polecenia na listę zadań do wykonania.

Autor: Michał Bojanowski

Ćwiczenia 24 (pl.sdacademy.designpatterns.interpreter)

- Stwórz interfejt Interpreter z metodą interpreter(String)
- Dodaj implementację interpretatora który jest w stanie zrozumieć operację na dwóch liczbach
  - dodawania np. ”2 + 3”
  - odejmowania np. ”3 - 1”
  - mnożenia, np. ”3 * 5”
  - dzielenia, np. ”5 / 2”
  - reszty z dzielenia, np. ”10 // 4”
  - potęgowania, np. ”2 ^ 3”

- Przetestuj swoją implementację w mainie
- Dodaj możliwość wykonywania wielu działań na raz (bez zachowania matematycznej kolejności wykonywania działań), np. ”3 * 7 / 2 // 3”

Autor: Michał Bojanowski

Zawartość dodatkowa

Autor: Michał Bojanowski

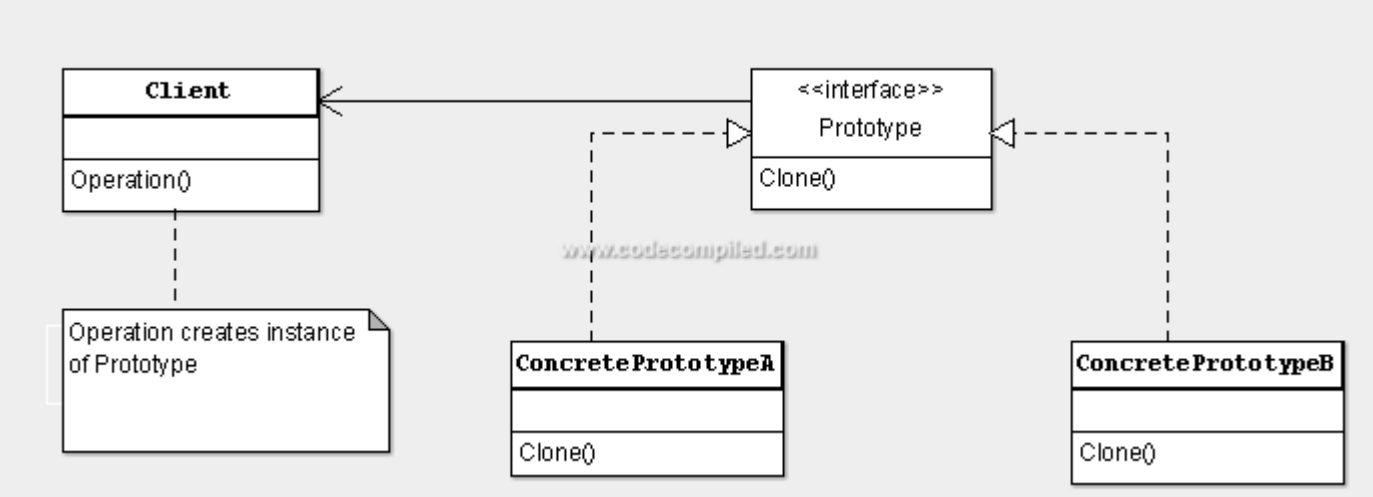
Prototype

Problem?
Chcemy stworzyć wiele obiektów tego samego typu, która część ze swoich pól ma zawsze taką samą wartość

Rozwiązanie?
Stworzyć prototyp tzn. klasę która kopiuje częściowo przygotowany obiekt

Autor: Michał Bojanowski

Prototype



Podczas używania wzorca Prototype weźmy pod uwagę **ilość** kopiowanych obiektów

Autor: Michał Bojanowski

Bridge

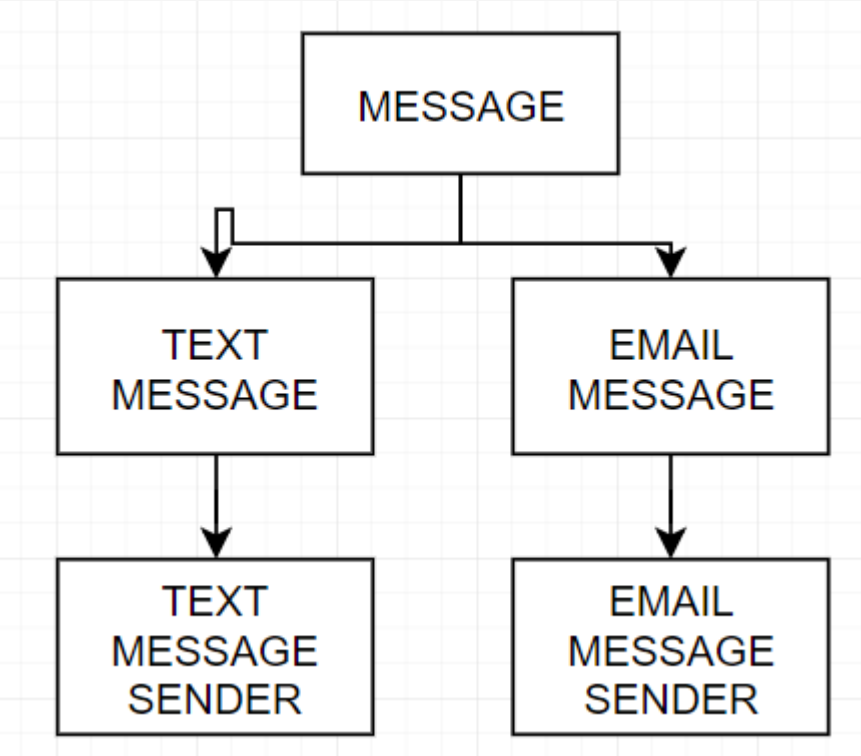
Uwaga! Często mylony z wzorcem Adapter

Bridge to wzorzec projektowy, który rozłącza abstrakcje i implementacje na dwie osobne hierarchie

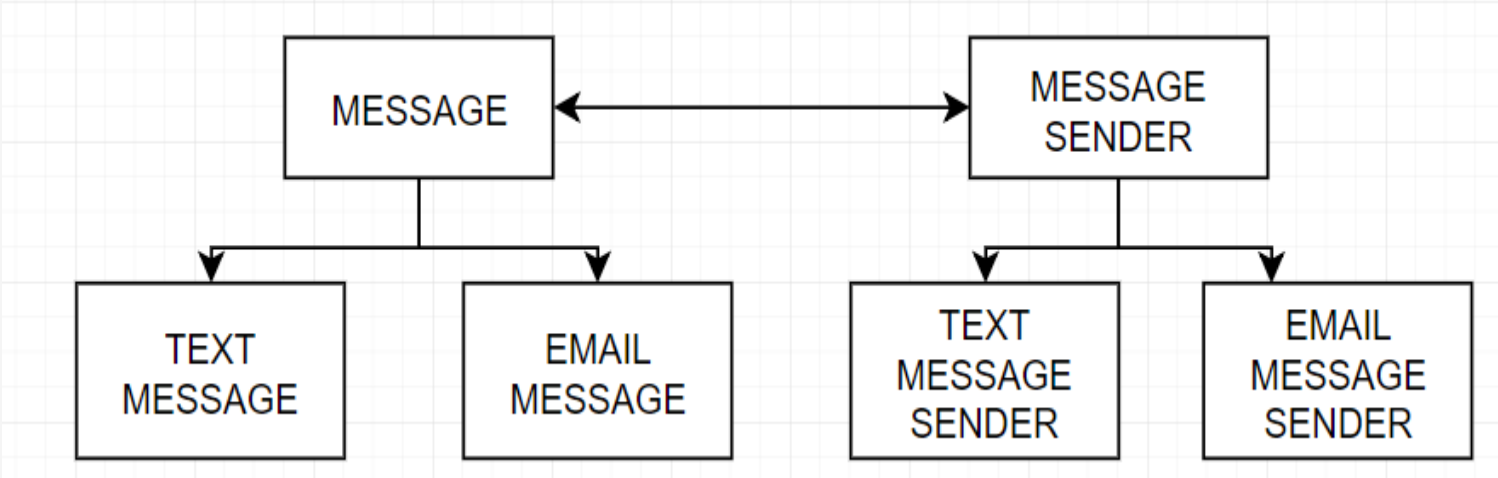
Autor: Michał Bojanowski

Bridge

BEZ BRIDGU



Z BRIDGEM



Autor: Michał Bojanowski

Flyweight

Problem?
Potrzebujemy stworzyć wiele obiektów...
Naprawdę wiele...
Tak dużo, że serwer może mieć za mało pamięci aby stworzyć ilość instancji jaką potrzebujemy. Wiele z tych obiektów jest do siebie podobnych.

Rozwiązanie?
Stworzyć **jeden** obiekt który zawiera dane wspólne dla wielu obiektów i jest dla nich referencją.
Obiekt docelowy zawiera referencję do danych wspólnego obiektu + ewentualnie dane potrzebne do rozróżnienia go od innych tego samego typu.

Autor: Michał Bojanowski

Flyweight

Przykład.
Tworzymy repozytorium reprezentujące listę samochdów w fabryce produkującą konkretny model samochodu (tzn. ma taki sam silnik, wyposażenie, skrzynię biegów i inne elementy mechaniczne)

Autor: Michał Bojanowski

Iterator

Problem?
Jak zapewnić sekwencyjny dostęp do obiektów które są zgrupowane w innym obiekcie?

Rozwiązanie
Stworzyć iterator czyli interfejs dający możliwość do poruszania po kontenerze.

Rodzaje iteratorów
- forward iterator
- backwards iterator
- bidirectional iterator

Autor: Michał Bojanowski