

Aplikacje Internetowe 1
mgr inż. Artur Karczmarczyk

Wzorce projektowe

S1: T-W-5

Agenda

- The Gang of Four
- Przypomnienie UML
- Wzorce behawioralne:
 - Strategia
- Wzorce kreacyjne:
 - Simple Factory
 - Factory Method
 - Abstract Factory
 - Lazy Initialization
 - Builder
 - Prototype
- Wzorce strukturalne:
 - Decorator
 - Adapter
 - Facade

Test

1. W UML implementację oznaczamy:
 - A. linią ciągłą i pustym trójkątem
 - B. linią przerywaną i pustym trójkątem
 - C. linią ciągłą i wypełnionym trójkątem
2. Definiuje rodzinę algorytmów, heremetyzuje każdy z nich i czyni je wymiennymi. Pozwala na dynamiczną zmianę algorytmu. Jaki to wzorzec?
 - A. Builder
 - B. Strategy
 - C. Facade
3. Dodaje funkcjonalność do klasy, nie zmieniając zachowania istniejących obiektów tej klasy. Pozwala na dodawanie zachowań w sposób dynamiczny. Jaki to wzorzec?
 - A. Strategy
 - B. Decorator
 - C. Simple Factory

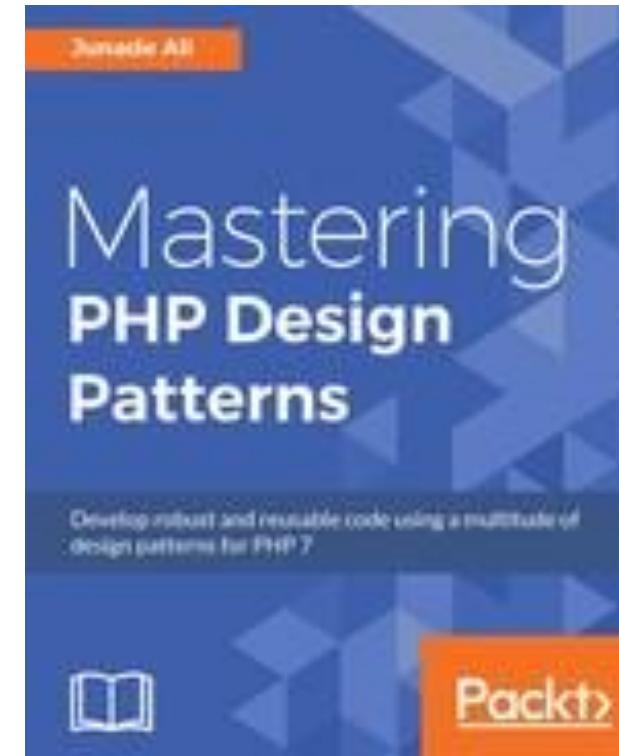
Literatura

- Head First Design Patterns, 2nd Edition
 - by Eric Freeman, Elisabeth Robson
 - Publisher: O'Reilly Media, Inc.
 - Release Date: December 2020
 - ISBN: 9781492078005



Literatura

- Mastering PHP Design Patterns
 - by Junade Ali
 - Publisher: Packt Publishing
 - Release Date: September 2016
 - ISBN: 9781785887130



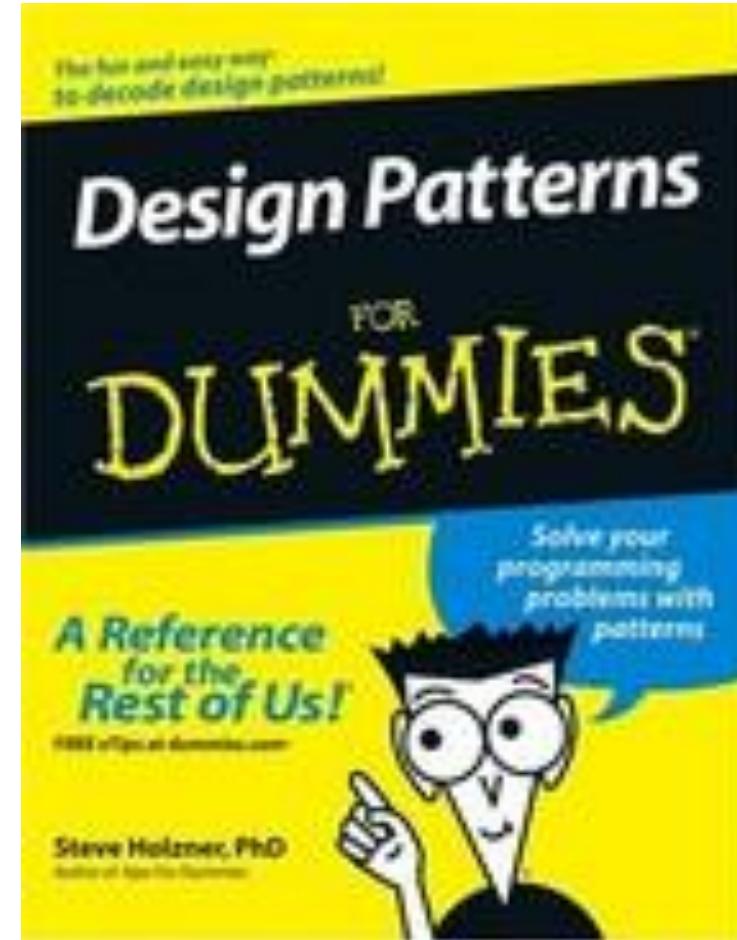
Literatura

- Mastering JavaScript Design Patterns - Second Edition
 - by Simon Timms
 - Publisher: Packt Publishing
 - Release Date: June 2016
 - ISBN: 9781785882166



Literatura

- Design Patterns For Dummies®
 - Publisher: Wiley
 - Release Date: May 2006
 - ISBN: 9780471798545



Czym są wzorce projektowe?

1977: A Pattern Language: Towns, Buildings, Construction

- Książka napisana przez:
 - Christopher Alexander
 - Sara Ishikawa
 - Murray Silverstein
- Opis języka do mówienia o wspólnych cechach projektowania.
- Wzorce:
 - *Elementy tego języka to byty zwane wzorcami. Każdy wzorzec opisuje problem, który pojawia się wielokrotnie w naszym środowisku, a następnie opisuje sedno rozwiązania tego problemu w taki sposób, że możesz użyć tego rozwiązania milion razy, nigdy nie robiąc tego w ten sam sposób dwa razy.*
 - Wzorce obejmowały takie tematy jak rozplanowanie miast, żeby zapewnić balans życia w mieście i na wsi; jak budować drogi, żeby uspokajać ruch na obszarach mieszkalnych.

1994: Gang of Four

- Zainspirowani językiem wzorców, czwórka autorów tworzy książkę:
 - Design Patterns: Elements of Reusable Object-Oriented Software
 - Erich Gamma
 - Richard Helm
 - Ralph Johnson
 - John Vlissides
- Książka przewana **Gang of Four book** - od 4 autorów.
- 23 wzorce do użycia w OOP, podzielone na 3 grupy...

Wzorce projektowe wg GoF

Kreacyjne

- do tworzenia obiektów
- do zarządzania cyklem życia obiektów

Behawioralne

- opisują wzajemne oddziaływanie obiektów między sobą

Strukturalne

- opisują sposoby dodawania funkcjonalności do istniejących obiektów

Więcej wzorców...

- Wzorce projektowe zawierają wskazówki dotyczące rozwiązywania typowych problemów.
- Oryginalna baza 23 wzorców znacznie się rozbudowała.
 - Na Wikipedii skategoryzowane wiele ponad 100 wzorców.
- Nie istnieją i nie mogą istnieć biblioteki wzorców.
 - Dlaczego?

V · T · E	Software design patterns	[hide]
Creational	Abstract factory · Builder · Dependency injection · Factory method · Lazy initialization · Multiton · Object pool · Prototype · RAII · Singleton	
Structural	Adapter · Bridge · Composite · Decorator · Delegation · Facade · Flyweight · Front controller · Marker interface · Module · Proxy · Twin	
Behavioral	Blackboard · Chain of responsibility · Command · Interpreter · Iterator · Mediator · Memento · Null object · Observer · Servant · Specification · State · Strategy · Template method · Visitor	
Functional	Monoid · Functor · Applicative · Monad · Comonad · Free monad · HOF · Currying · Function composition · Closure · Generator	
Concurrency	Active object · Actor · Balking · Barrier · Binding properties · Coroutine · Compute kernel · Double-checked locking · Event-based asynchronous · Fiber · Futex · Futures and promises · Guarded suspension · Immutable object · Join · Lock · Messaging · Monitor · Nuclear · Proactor · Reactor · Read write lock · Scheduler · STM · Thread pool · Thread-local storage	
Architectural	ADR · Active record · Broker · Client-server · CBD · DAO · DTO · DDD · ECB · ECS · EDA · Front controller · Identity map · Interceptor · Implicit invocation · Inversion of control · Model 2 · MOM · Microservices · MVA · MVC · MVP · MVVM · Monolithic · Multitier · Naked objects · ORB · P2P · Publish-subscribe · PAC · REST · SOA · Service locator · Specification	
Cloud Distributed	Ambassador · Anti-Corruption Layer · Bulkhead · Cache-Aside · Circuit Breaker · CQRS · Compensating Transaction · Competing Consumers · Compute Resource Consolidation · Event Sourcing · External Configuration Store · Federated Identity · Gatekeeper · Index Table · Leader Election · MapReduce · Materialized View · Pipes · Filters · Priority Queue · Publisher-Subscriber · Queue-Based Load Leveling · Retry · Scheduler Agent Supervisor · Sharding · Sidecar · Strangler · Throttling · Valet Key	
Other	Business delegate · Composite entity · Intercepting filter · Lazy loading · Mangler · Mock object · Type tunnel · Method chaining	
Books	<i>Design Patterns</i> · <i>Enterprise Integration Patterns</i> · <i>Code Complete</i> · <i>POSA</i>	
People	Christopher Alexander · Erich Gamma · Ralph Johnson · John Vlissides · Grady Booch · Kent Beck · Ward Cunningham · Martin Fowler · Robert Martin · Jim Coplien · Douglas Schmidt · Linda Rising	
Communities	The Hillside Group · The Portland Pattern Repository	

Formalna definicja wzorca

- Wzorzec to rozwiązanie problemu w kontekście.
- Kontekst
 - sytuacja, w której wzorzec ma zastosowanie
 - sytuacja taka powinna się powtarzać
- Problem
 - dotyczy celu, który jest do rozwiązania w danym kontekście
 - uwzględnia wszystkie potencjalne ograniczenia w danym kontekście
- Rozwiązanie
 - ogólne podejście, które każdy może zastosować
 - realizuje postawione cele w zadanym zestawie ograniczeń

UML

Przypomnienie diagramów klas

Klasa i właściwości publiczne

Klasa

+ polePubliczne: string

Właściwości prywatne

Klasa

+ polePubliczne: string

- polePrywatne: string

Właściwości chronione

Klasa

+ polePubliczne: string

poleChronione: string

- polePrywatne: string

Metody

Klasa

+ polePubliczne: string

poleChronione: string

- polePrywatne: string

+ metodaPubliczna(string, int): string

Właściwości / metody abstrakcyjne

Klasa

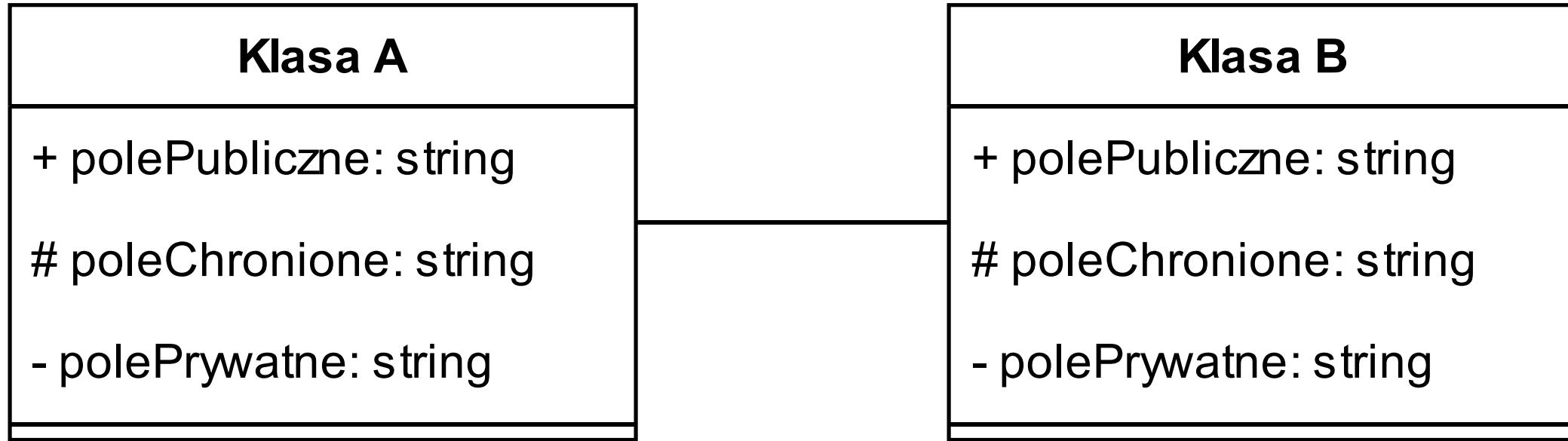
+ polePubliczne: string

poleChronione: string

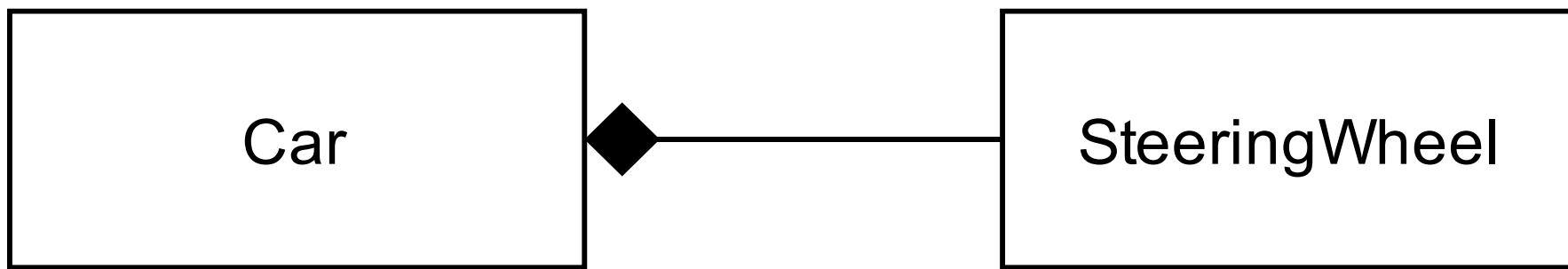
- polePrywatne: string

+ metodaAbstrakcyjna(string, int): string

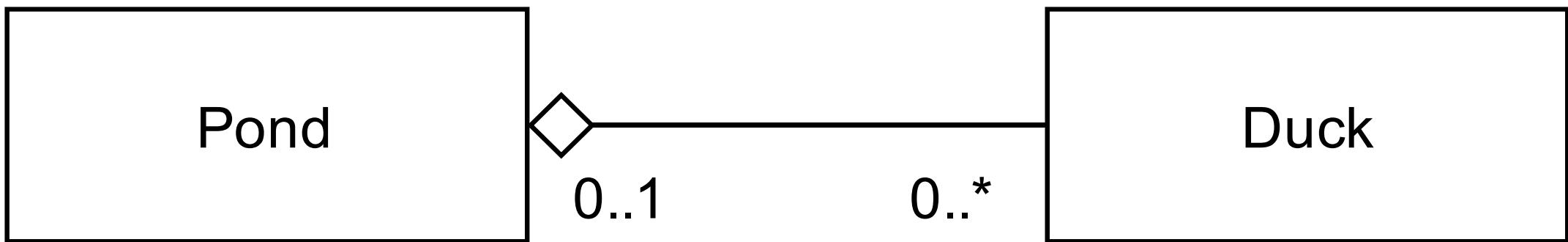
Asocjacja



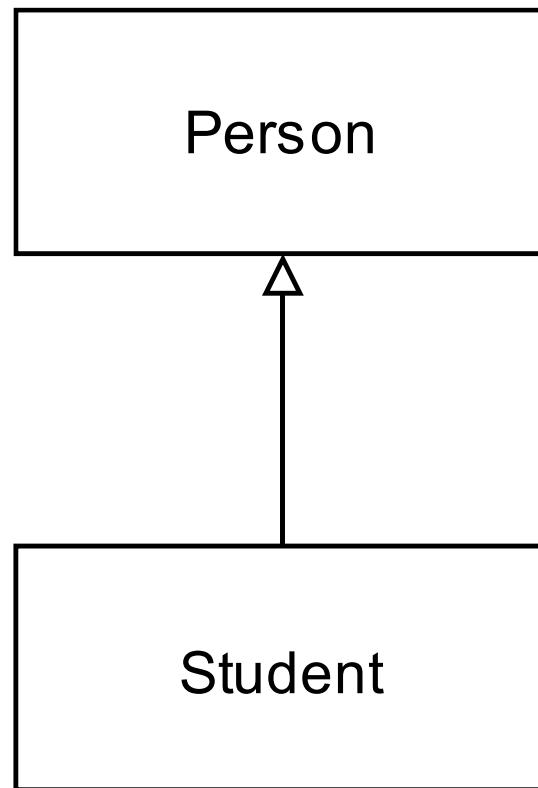
Kompozycja



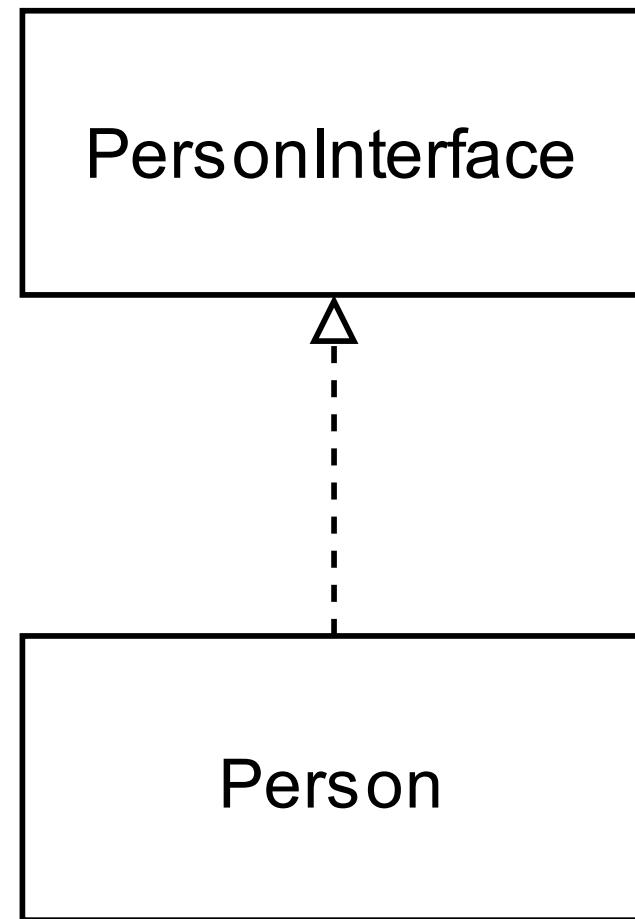
Agregacja



Dziedziczenie



Implementacja



Wyzwanie

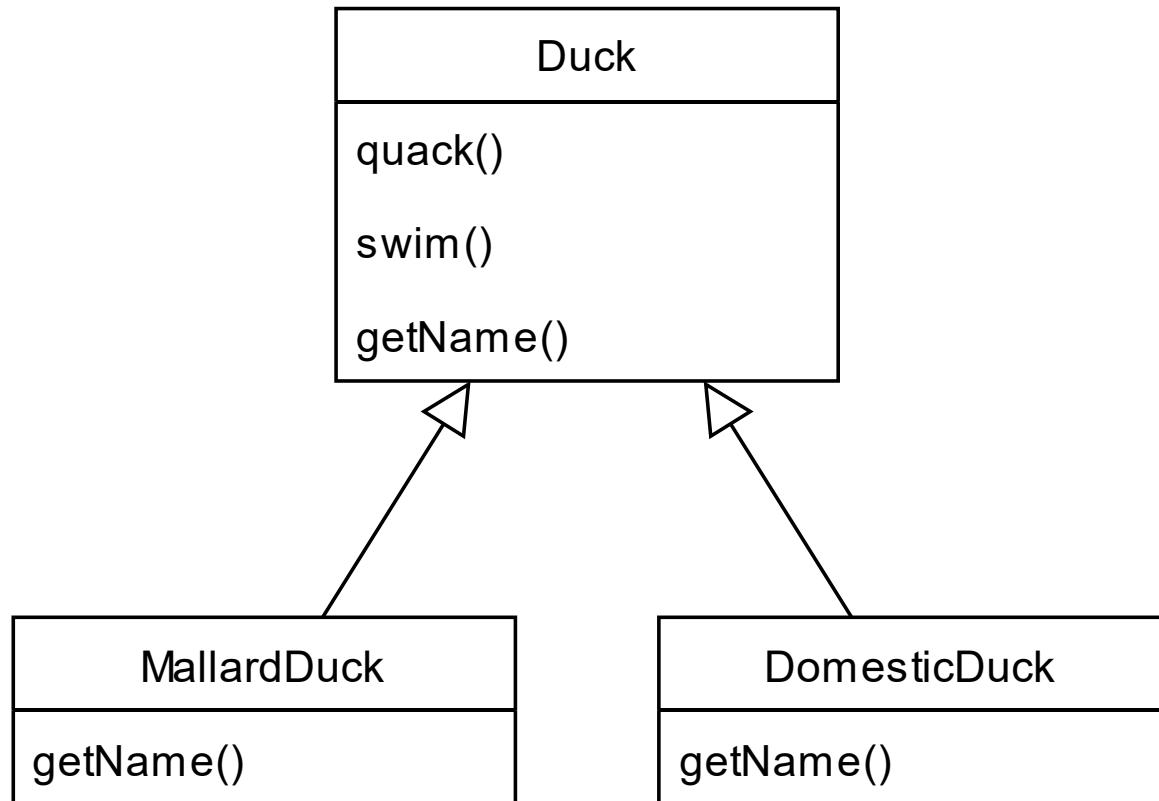
- Narysować diagram klas UML:
- klasa Duck, z publicznymi metodami quack() i swim()
- klasy MallardDuck (kaczka krzyżówka) i DomesticDuck (kaczka domowa), dziedziczące po klasie Duck.

Strategia

Wzorzec behawioralny

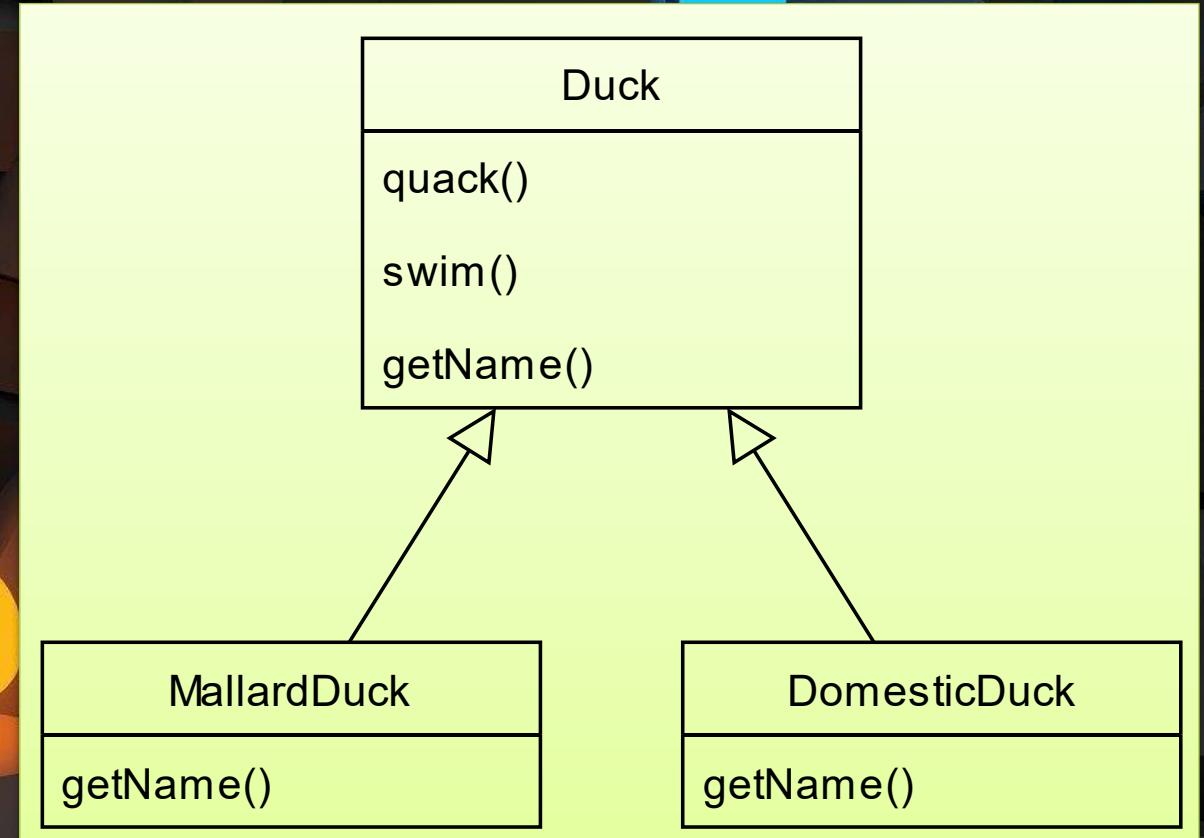
Symulator stawu

- Zaprojektowano grę – symulator stawu.
- Bohaterami gry są kaczki.
- Kaczki mogą pływać (swim) i kwakać (quack).
- Na obecnym etapie obsługiwane są dwa rodzaje kaczek:
 - kaczka krzyżówka (mallard duck)
 - kaczka domowa (domestic duck)
- Jak to zaimplementować?



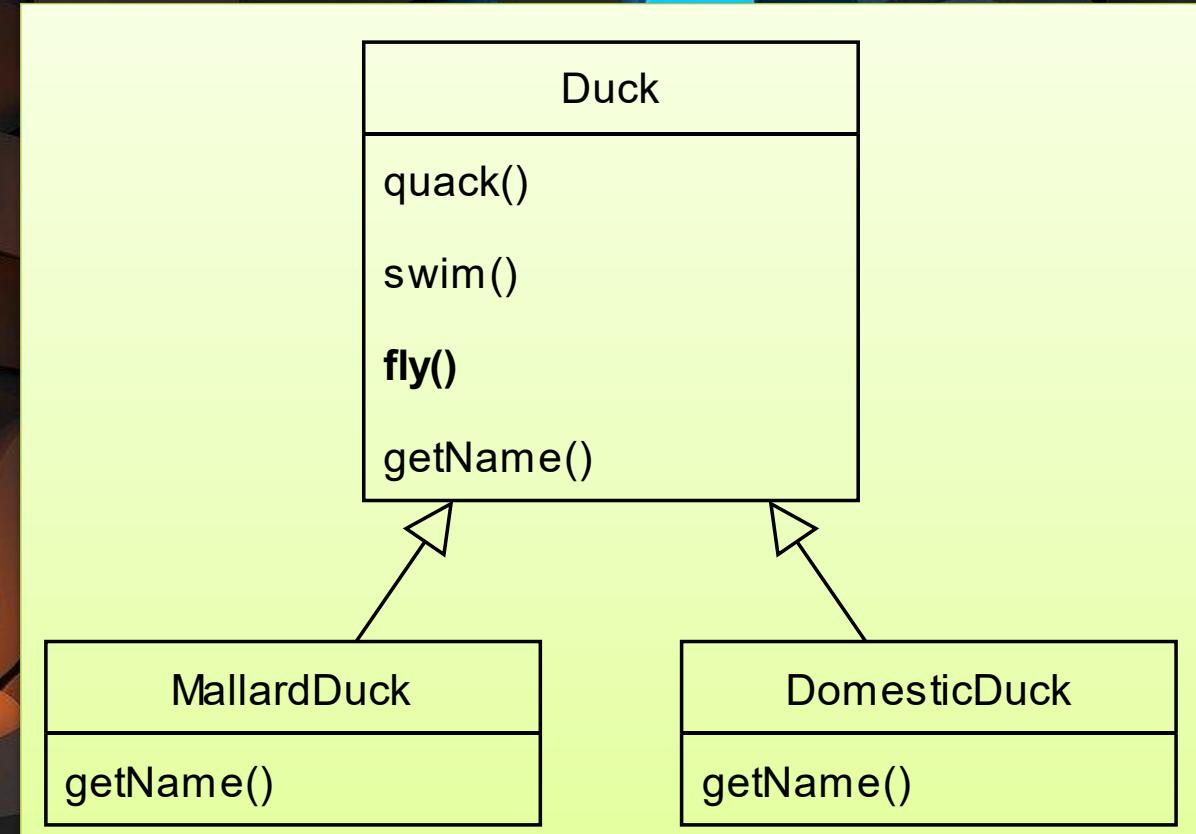
Wyzwanie

- Zaimplementować klasy do obsługi kaczki krzyżówki i kaczki domowej.
- Zastosować dziedziczenie.



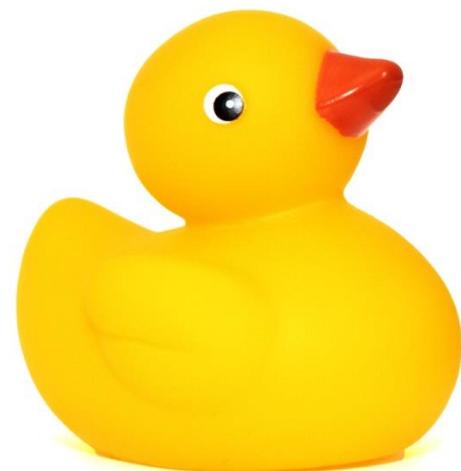
Wyzwanie

- Wykorzystując paradygmat obiektowy, dodać do kaczek funkcjonalność latania – fly().



Problemy...

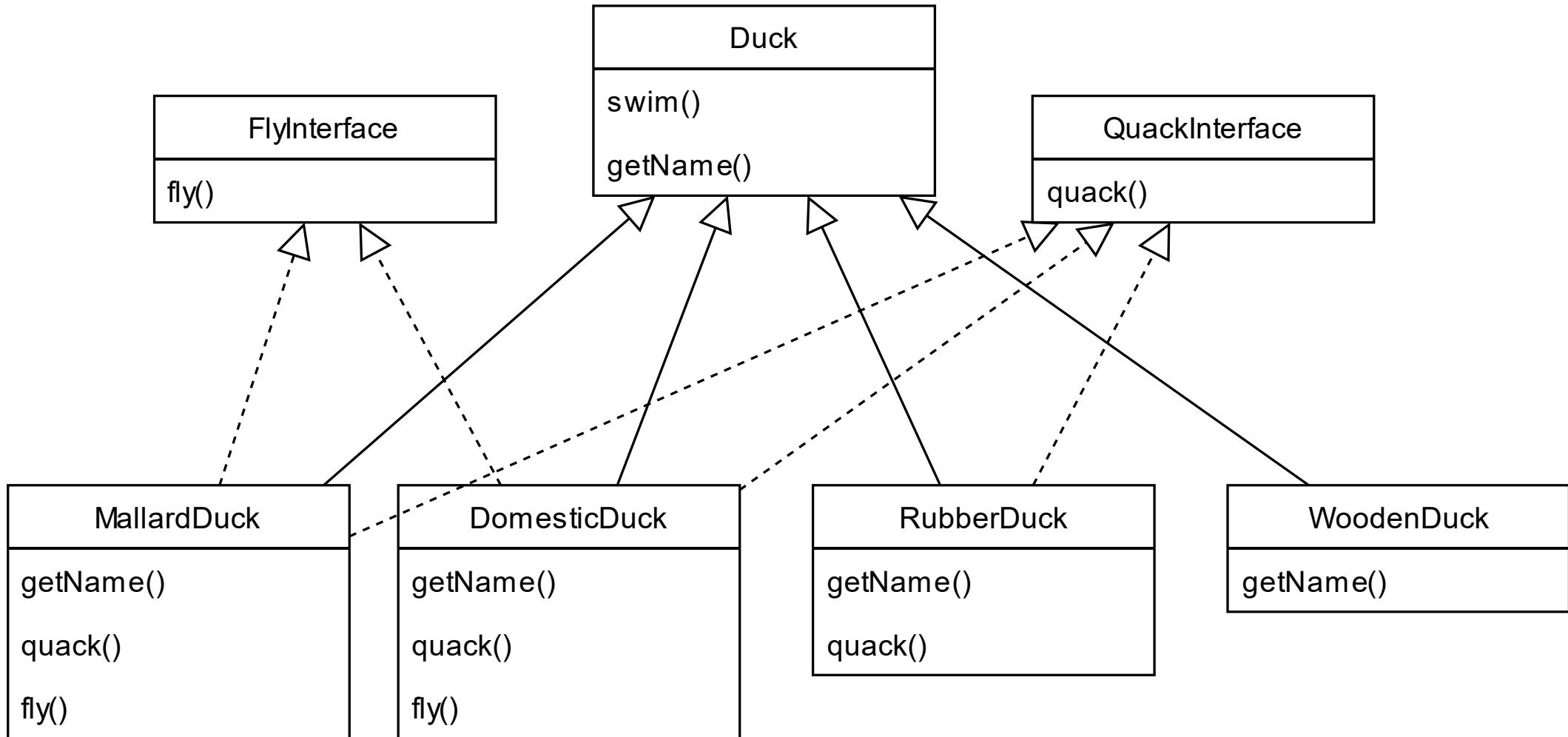
- Do symulatora dodano nową klasę RubberDuck.
 - Nadpisano metodę quack() żeby robiła „squeak”.
 - Czy coś się wydarzyło?...
-
- Gumowe kaczki potrafią latać!
 - Jak to rozwiązać?



A co z drewnianymi kaczkami?

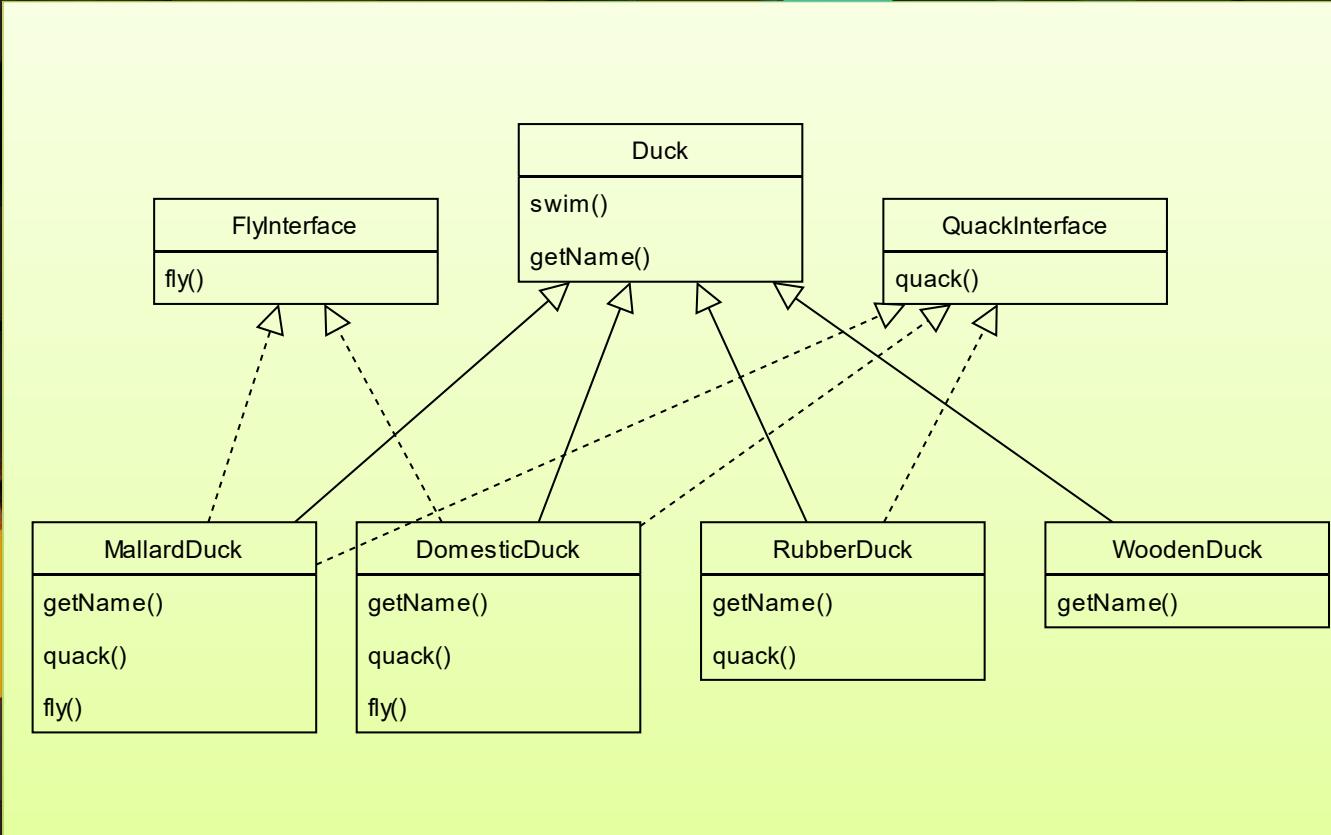


To może interfejs?

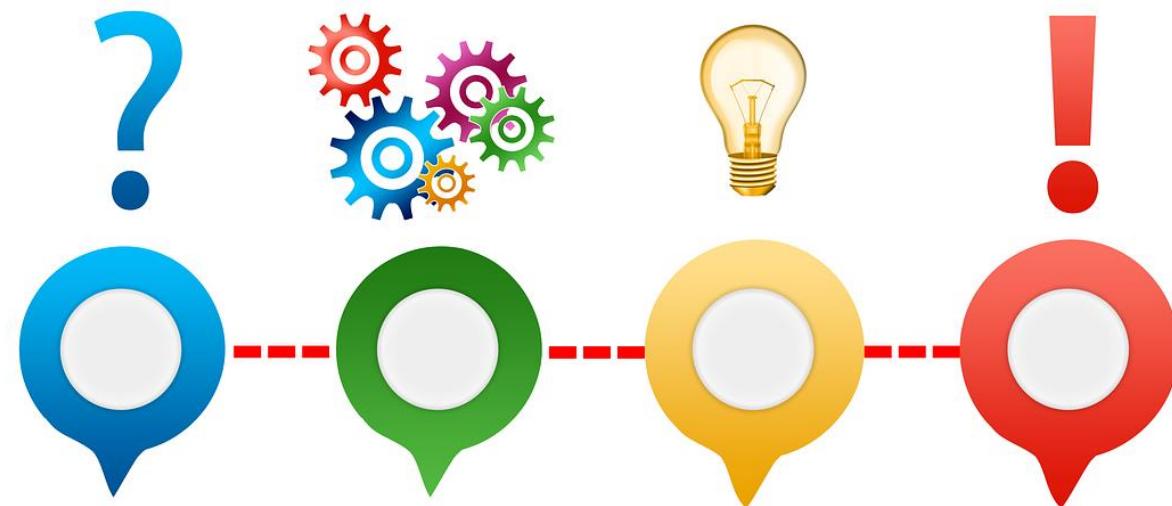


Wyzwanie

- Wykorzystać interfejsy do implementacji 4 typów kaczek:
 - MallardDuck
 - DomesticDuck
 - RubberDuck
 - WoodenDuck



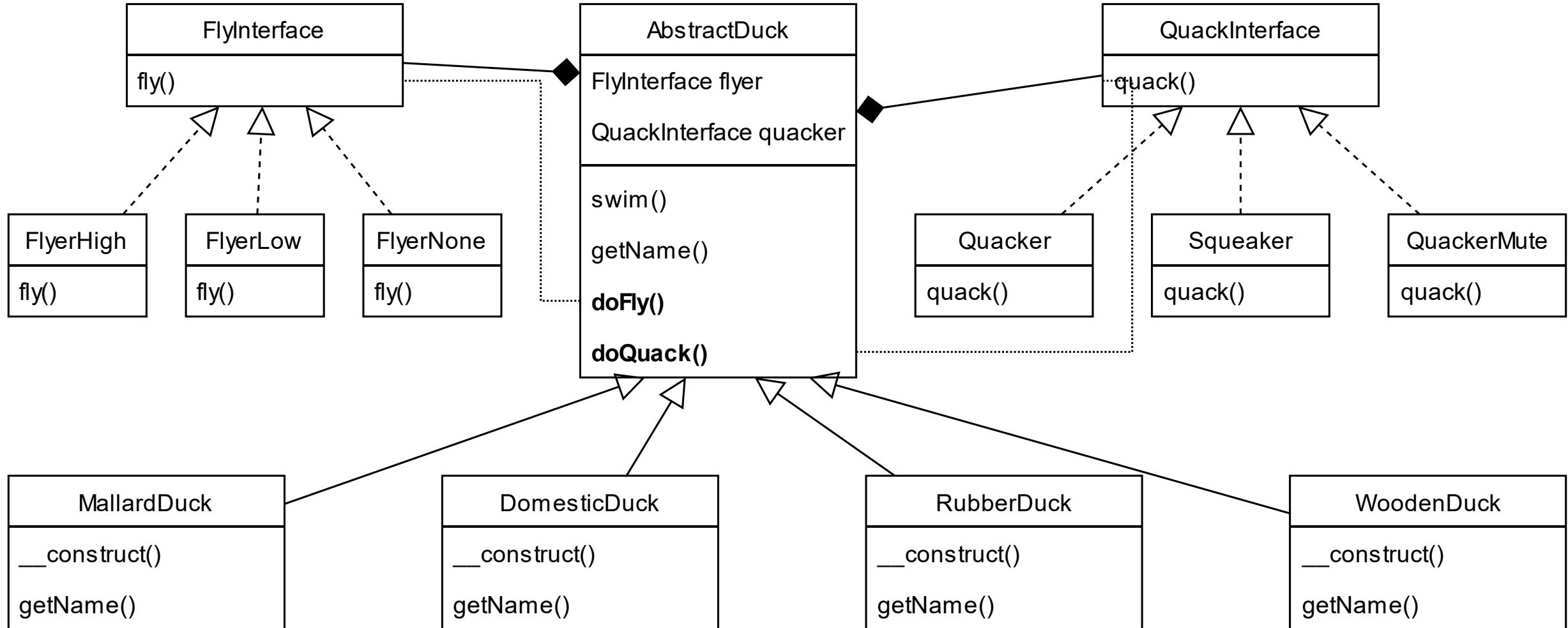
Jak uniknąć kopiowania kodu?



Oddzielenie zmian

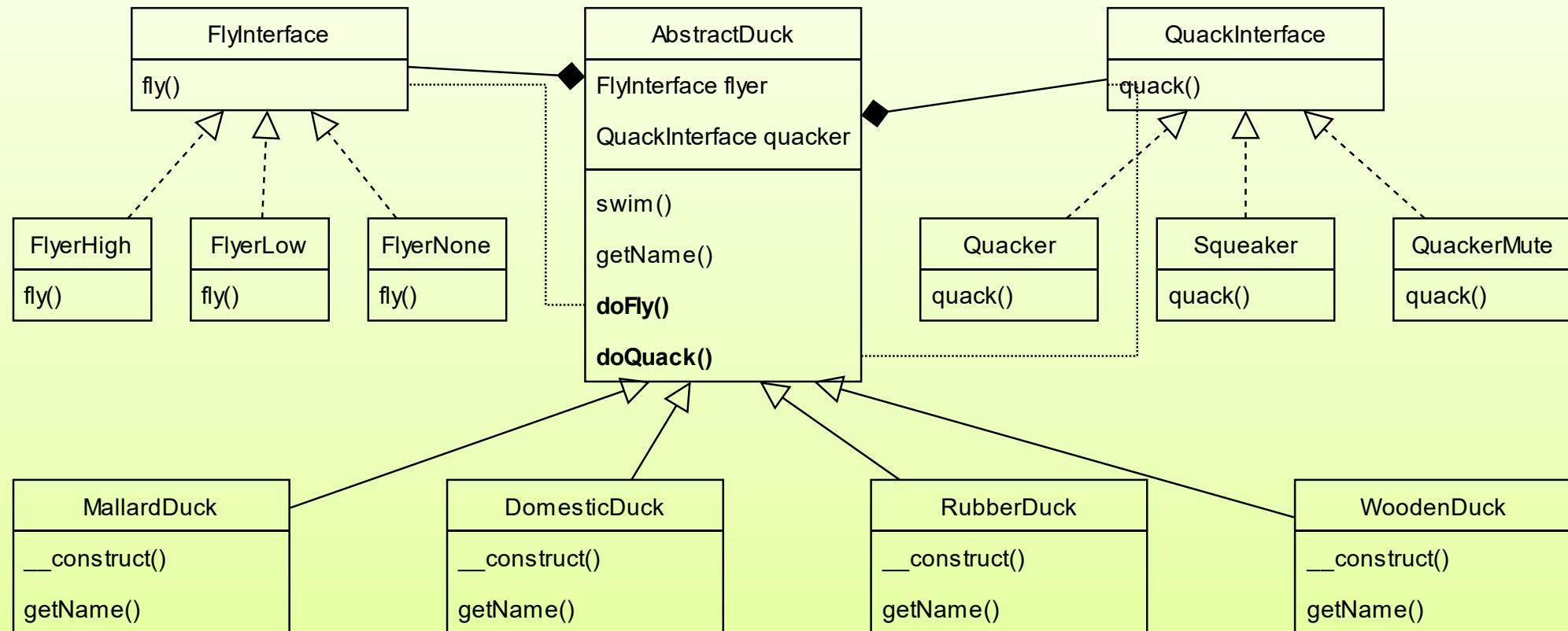
- Dobrą praktyką rozdzielenie:
 - tego co jest stałe
 - od tego co będzie się zmieniać.
- Stałe:
 - większość klasy Duck
 - metoda getName()
 - metoda swim()
- Zmienne:
 - metoda quack()
 - metoda fly()
- Metody quack() i fly() opakujemy do osobnych klas opisujących zachowanie.
- Implementacji kwakania i latania może być wiele...
 - zastosujemy interfejs!

Oddzielne klasy dla zachowań



Wyzwanie

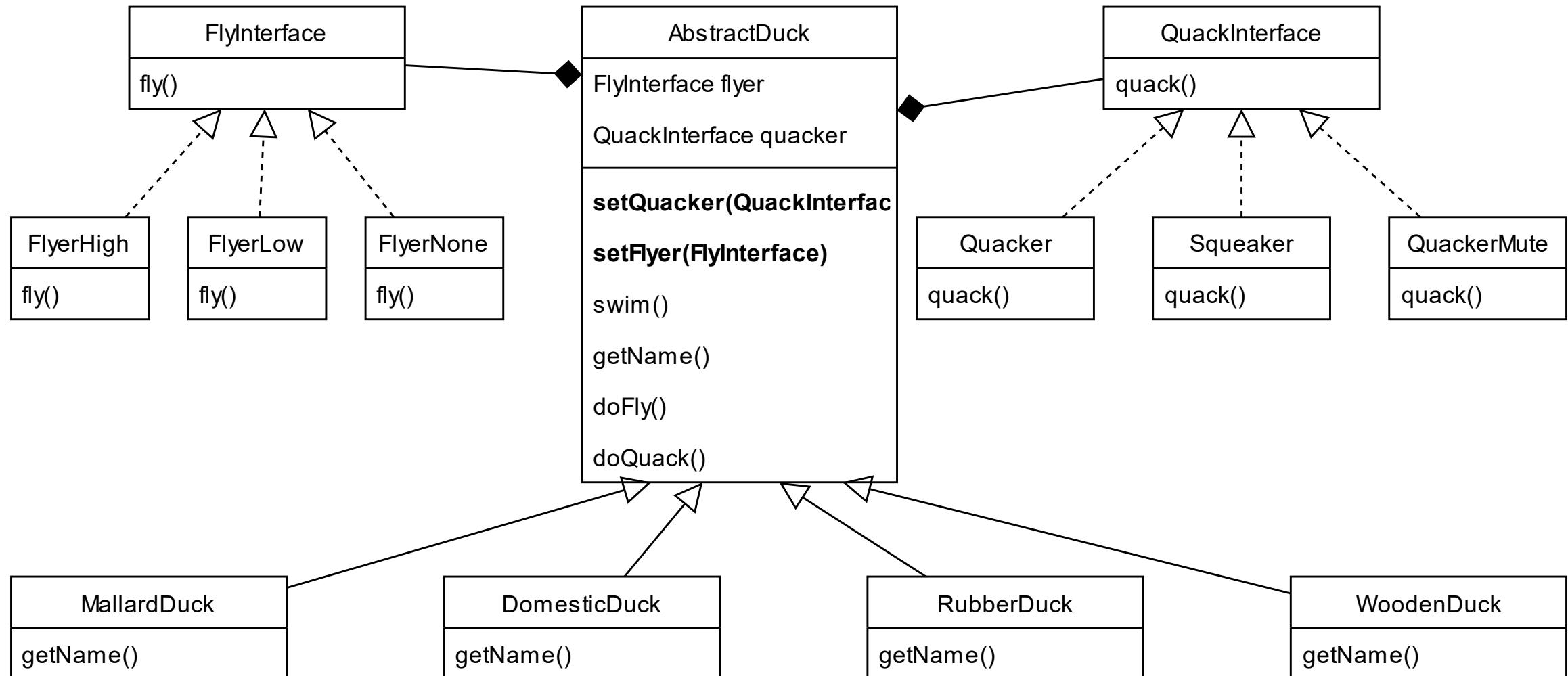
- Zaimplementować poniższe:



Wyzwanie

- Czy zaimplementowane podejście jest lepsze od użycia traitów?
- Czy traity można podmieniać w czasie wykonania (runtime)?
- Czy w zaimplementowanym podejściu można zmieniać zachowania w czasie wykonania (runtime)?
- Dodajmy taką możliwość!
 - nowa metoda `setFlyer(FlyInterface $flyer)`
 - nowa metoda `setQuacker(QuackInterface $squacker)`
 - usunięcie wpisów „na sztywno” z konstruktora

Podmiana zachowań w trakcie wykonania



Wyzwanie

- Jak zakodować scenariusz, w którym kaczka krzyżówka (MallardDuck) ulegnie kontuzji i nie będzie mogła latać?
- Jak zakodować scenariusz, w którym gumowa kaczka (RubberDuck) ulegnie awarii i nie będzie więcej piszczeć?

Wzorzec strategia

- Wzorzec strategii:
 - definiuje rodzinę algorytmów,
 - hermetyzuje każdy z nich
 - i czyni je wymiennymi.
- Strategia pozwala na zmianę algorytmu niezależnie od klientów, którzy go używają.

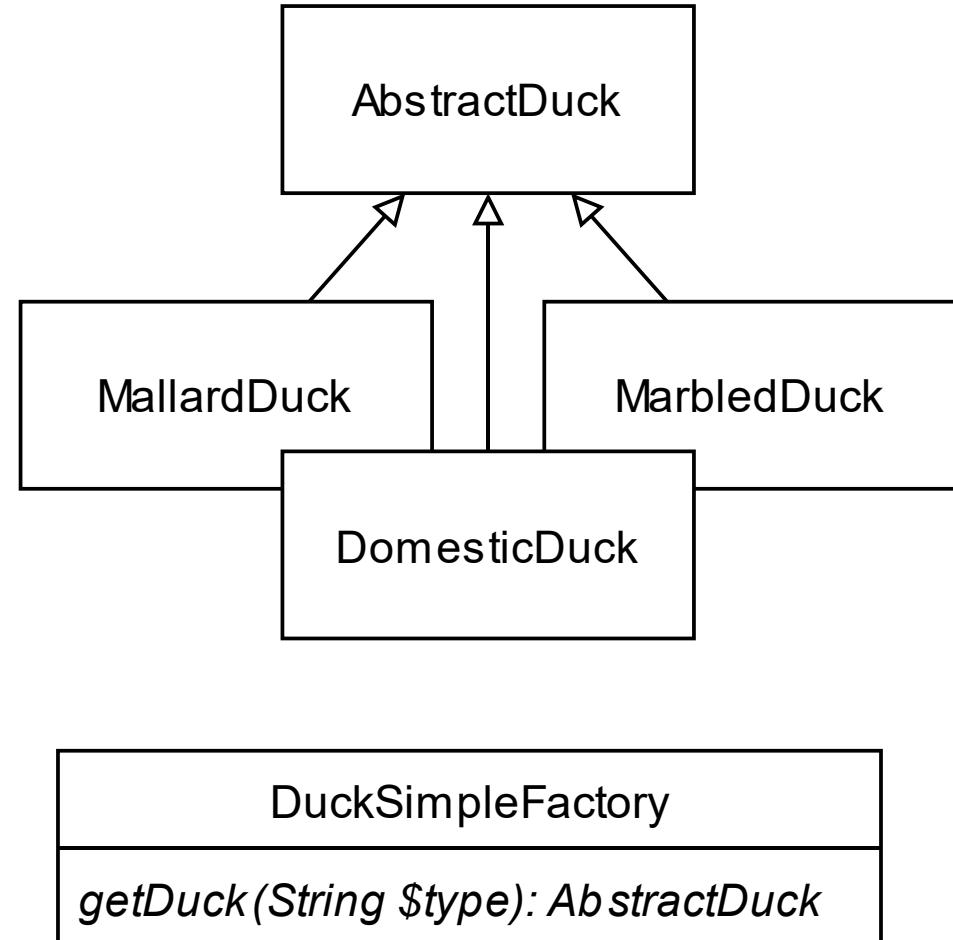
Podstawowe zasady projektowe

- Identyfikuj w aplikacji to co zmienne i oddzielaj to od tego co stałe.
- Programuj do interfejsu, a nie do implementacji.
 - w klasach QuackInterface a nie Quacker / QuackerMute / Squeaker
- Preferuj kompozycję ponad dziedziczenie.

Wzorce kreacyjne

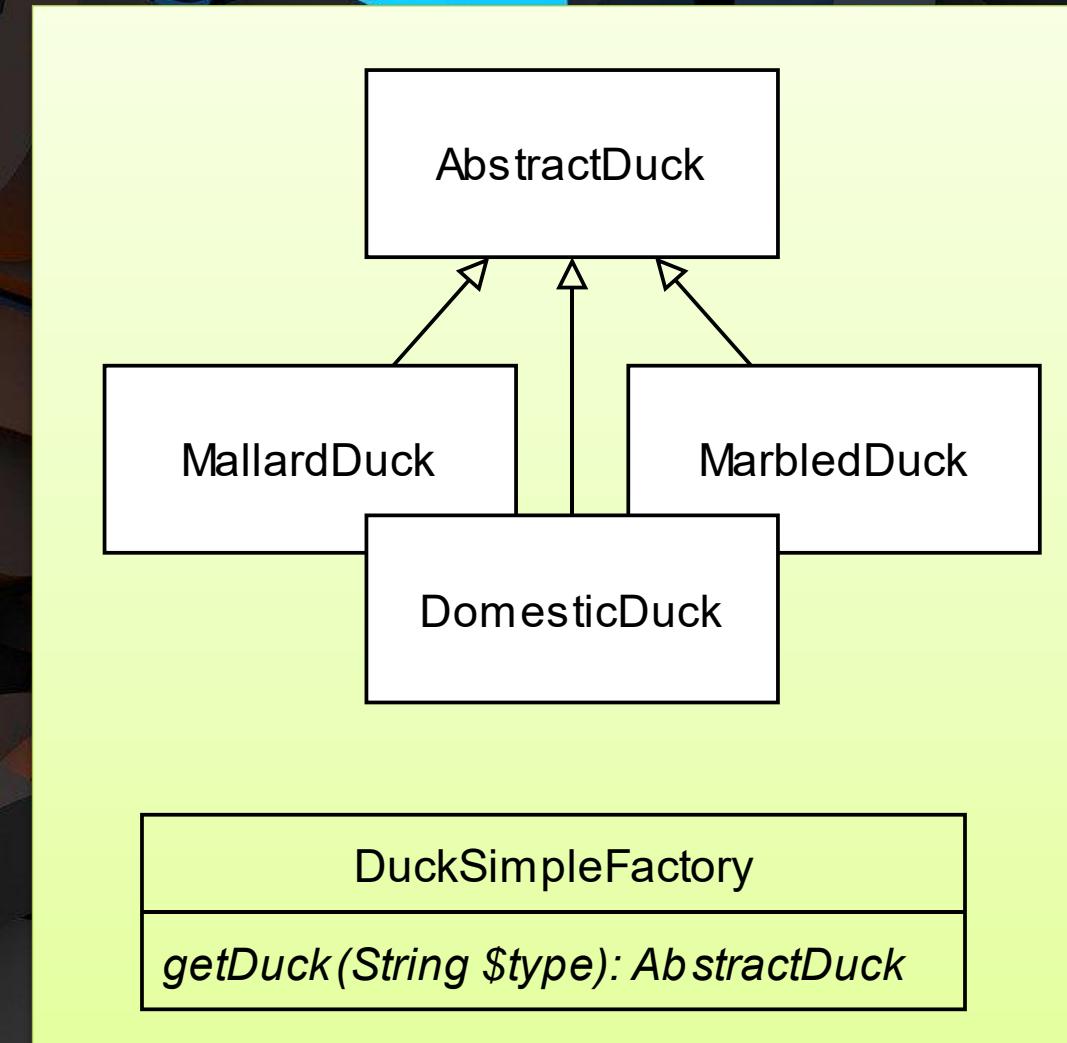
Simple Factory (prosta fabryka)

- Prosta fabryka nie do końca jest wzorcem projektowym.
 - Raczej idiom programowania.
 - Bardzo często używane podejście.
-
- Klasa abstrakcyjna grupująca różne klasy konkretne.
 - Osobna klasa, posiadająca metodę statyczną zwracającą obiekt, w zależności od zadanego typu.



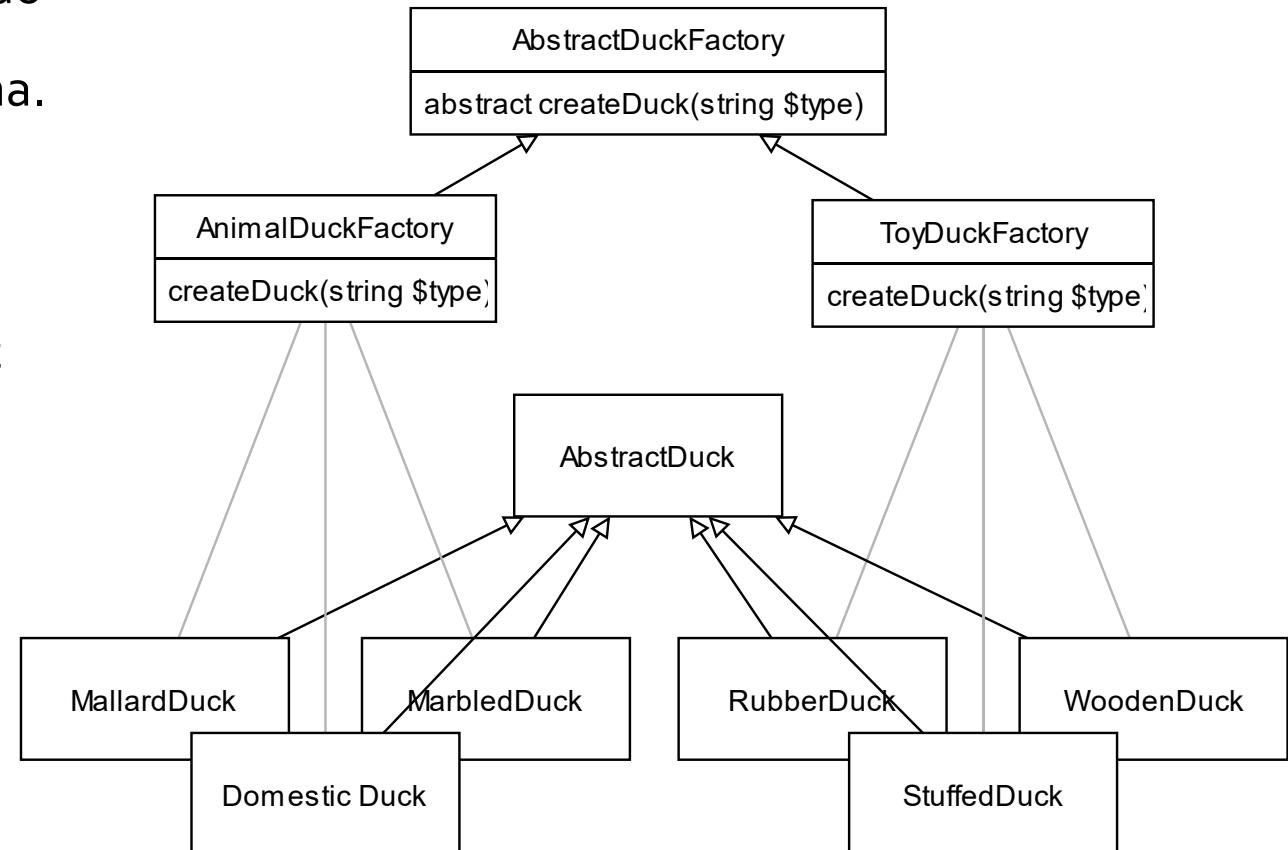
Wyzwanie

- Zbudować prostą fabrykę DuckSimpleFactory::getDuck().



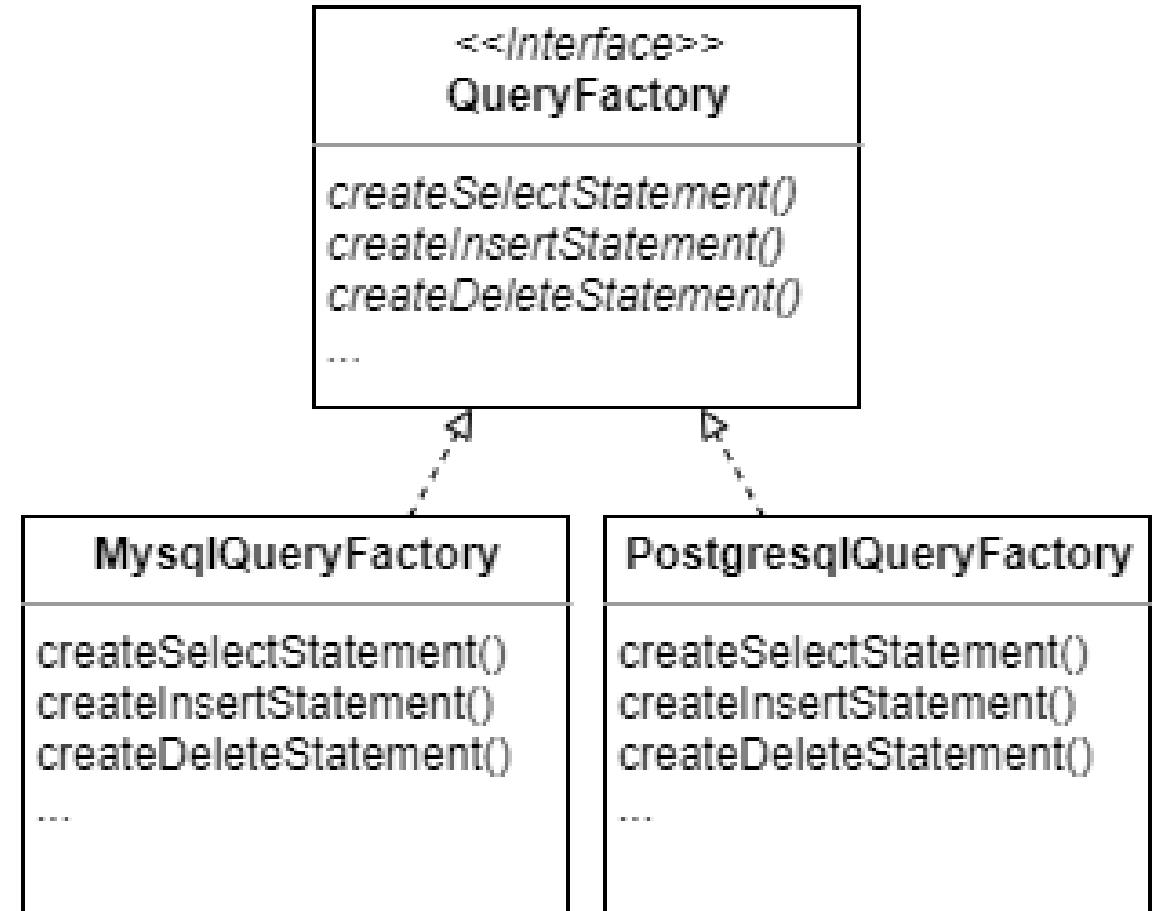
Factory Method (metoda fabrykująca)

- Wzorzec Factory Method definiuje interfejs do tworzenia obiektu, ale pozwala podklasom zdecydować która klasa ma zostać utworzona.
- Możemy mieć wiele fabryk zamiast jednej.
 - pizza włoska / amerykańska
 - kaczki zwierzęta / zabawki
- Wzorzec Factory Method pozwala grupować klasy i przypisywać do nich odrębne fabryki.
- W przypadku konieczności dodania nowej grupy klas, można dodać nową fabrykę
 - brak konieczności modyfikowania kodu
- Zasada projektowa:
 - klasy powinny być otwarte na rozszerzanie, ale zamknięte na modyfikację



Abstract Factory (fabryka abstrakcyjna)

- Wzorzec Abstract Factory udostępnia interfejs do tworzenia rodzin powiązanych lub zależnych obiektów
- Bez wskazywania konkretnych klas.
- Klient nie jest sprzężony z konkretnymi klasami produktów (decoupled).



Lazy Initialization (leniwe inicjowanie)

- Wzorzec polega na opóźnieniu:
 - tworzenia obiektu
 - obliczania wartości
 - przeprowadzania innych kosztownych operacji
 - aż do momentu zapotrzebowania na nie.
- I am User 1, fetched at 01:16:57.
 - I am User 2, fetched at 01:16:58.
 - I am User 3, fetched at 01:16:59.
 - I am User 1, fetched at 01:16:57.
 - I am User 2, fetched at 01:16:58.
 - I am User 3, fetched at 01:16:59.

Builder (budowniczy)

- Rozwiązuje problem teleskopowego konstruktora.
- Teleskopowy konstruktor:
 - liczba argumentów zawartych w konstruktorze rośnie
 - do tego stopnia, że ciężko użyć
 - do tego stopnia, że ciężko policzyć, który argument co oznacza
- Wzorzec polega na enkapsulacji tworzenia obiektu i zleceniu budowy innemu obiekowi.
- Pozwala na tworzenie obiektu w wielu krokach, „na raty”
 - fabryki są 1-etapowe

Wyzwanie

- Dana jest klasa User.
- Konstruktor:
 - `__construct($from, $to, $subject, $body, $cc = null, $bcc = null, $replyTo = null)`
- Wysłać maila od A do B z tematem i treścią.
- Wysłać maila od A do A z tematem i treścią, BCC do B.
- Wysłać maila od A do B z tematem i treścią, ale odpowiedź do C.
- Uprościć proces wzorcem Builder.

Prototype (prototyp)

- Pozwala tworzyć nowe instancje poprzez wykorzystanie istniejących obiektów.
 - W PHP realizowane z wykorzystaniem `clone()`.
 - W JS realizowane poprzez dostęp do prototypów.
-
- Upraszcza tworzenie nowych obiektów o dużej liczbie wspólnych pól.
 - Zwiększa wydajność, jeśli zwykły konstruktor zajmuje dużo czasu.

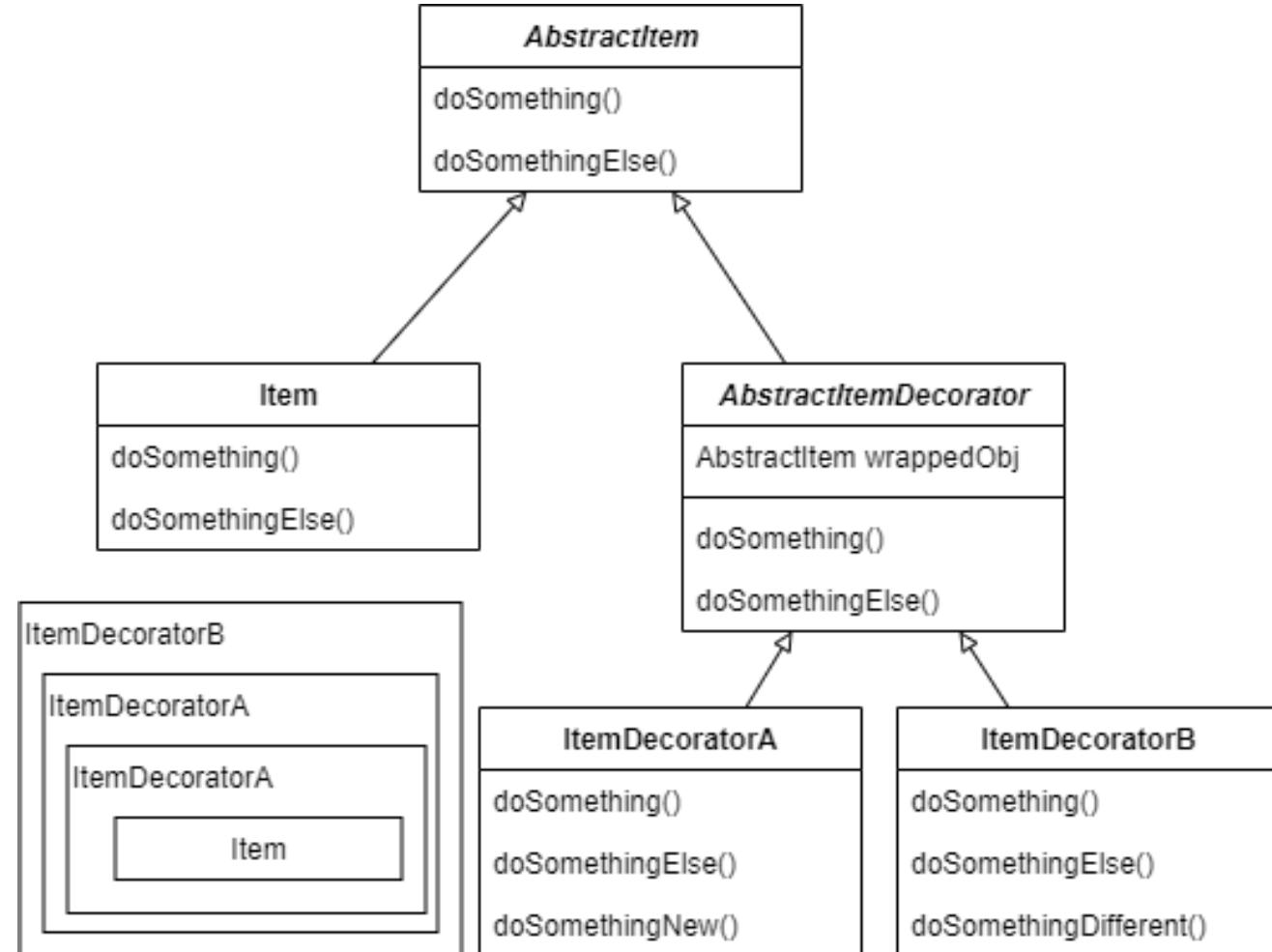
Wyzwanie

- Zastosuj wzorzec Prototype do przyśpieszenia i uproszczenia tworzenia wielu instancji klasy Student.

Wzorce strukturalne

Decorator (dekorator)

- Dodaje funkcjonalność do klasy.
 - Nie zmienia zachowania istniejących obiektów tej samej klasy.
 - Pozwala na dodawanie zachowań w sposób dynamiczny.
 - Elastyczna alternatywa do rozszerzania klas poprzez dziedziczenie.
-
- Zasada projektowa:
 - klasy powinny być otwarte na rozszerzanie, ale zamknięte na modyfikację

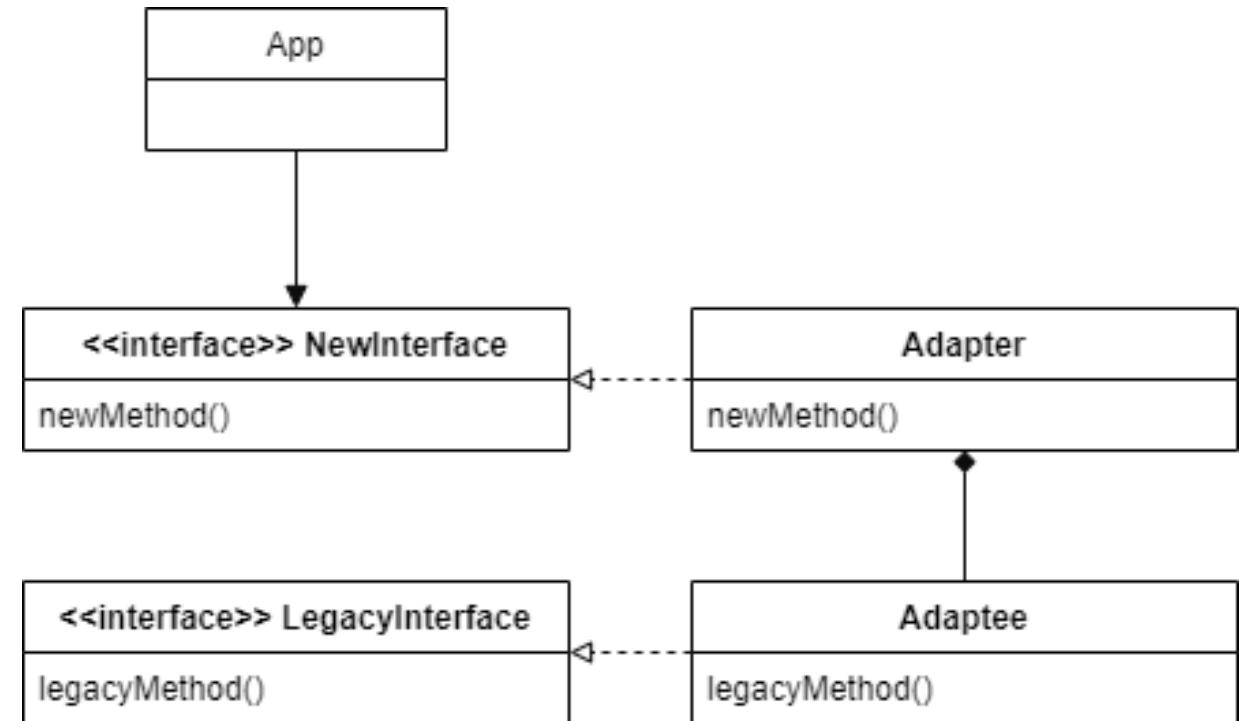


Wyzwanie

- Zastosować wzorzec Decorator do zbudowania:
 - miecza stalowego (cena 300/100, uszkodzenia 50)
 - miecza stalowego z zaklęciem (cena 300/100, uszkodzenia 60)
 - miecza stalowego z runem (cena 800/200, uszkodzenia 70)
 - miecza stalowego z runem i zaklęciem (cena 800/200, uszkodzenia 80)
- Nadać mieczowi w trakcie wykonywania programu nazwę
 - „Księżycowy Rzeźnik”

Adapter (przejściówka ;))

- Pozwala na użycie istniejących klas z interfejsem, do którego nie pasują.
- Pozwala na wykorzystanie ze sobą klas które na co dzień do siebie nie pasują.
- Nie wymaga modyfikacji kodu żadnej z klas.
- Klient i klasa połączone adapterem nie wiedzą o sobie.

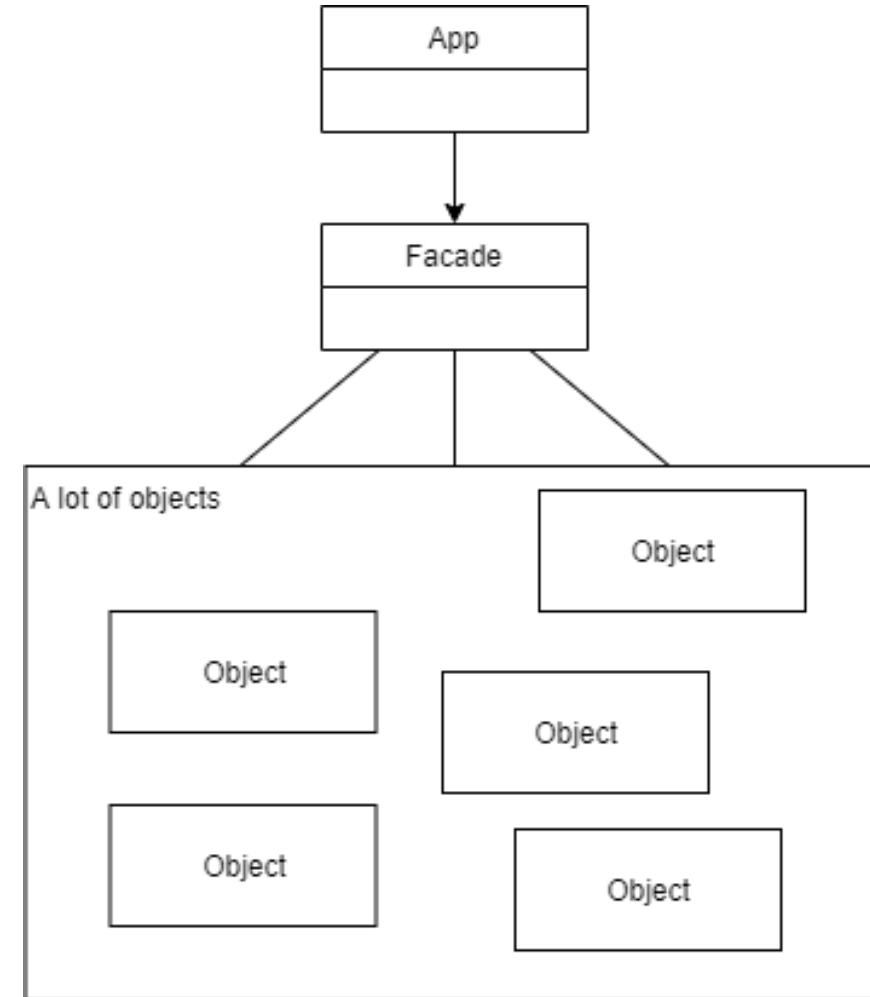


Wyzwanie

- Zastosować wzorzec Adapter w celu umożliwienia spójnego wykorzystania klas implementujących interfejsy:
 - ThrowableInterface
 - CuttableInterface
 - ShootableInterface

Façade (fasada)

- Stanowi prosty interfejs do złożonego systemu.
- Wzorzec udostępnia pojedynczą klasę, która sama tworzy instancje wielu różnych obiektów.
- Sterowanie obiekttami za pomocą prostych metod fasady.
- Możliwość dostępu ręcznego do poszczególnych obiektów.



Wyzwanie

- Wykorzystać wzorzec Facade do sterowania nastrojem świątecznym.
- Klasy:
 - ChristmasLights
 - LivingRoomLights
 - GingerAromatizer
 - Radio
- Fasada:
 - ChristmasModeFacade

Porównanie

Decorator

- Nie zmienia interfejsu.
- Dodaje odpowiedzialności dla klasy.

Adapter

- Konwertuje jeden interfejs na inny.
- Nie zmienia odpowiedzialności.

Facade

- Upraszczia interfejs.

Podsumowanie

- The Gang of Four
- Przypomnienie UML
- Wzorce behawioralne:
 - Strategia
- Wzorce kreacyjne:
 - Simple Factory
 - Factory Method
 - Abstract Factory
 - Lazy Initialization
 - Builder
 - Prototype
- Wzorce strukturalne:
 - Decorator
 - Adapter
 - Facade

Test

1. W UML implementację oznaczamy:
 - A. linią ciągłą i pustym trójkątem
 - B. linią przerywaną i pustym trójkątem
 - C. linią ciągłą i wypełnionym trójkątem
2. Definiuje rodzinę algorytmów, heremetyzuje każdy z nich i czyni je wymiennymi. Pozwala na dynamiczną zmianę algorytmu. Jaki to wzorzec?
 - A. Builder
 - B. Strategy
 - C. Facade
3. Dodaje funkcjonalność do klasy, nie zmieniając zachowania istniejących obiektów tej klasy. Pozwala na dodawanie zachowań w sposób dynamiczny. Jaki to wzorzec?
 - A. Strategy
 - B. Decorator
 - C. Simple Factory

Odpowiedzi

1. W UML implementację oznaczamy:
 - A. linią ciągłą i pustym trójkątem
 - B. **linią przerywaną i pustym trójkątem**
 - C. linią ciągłą i wypełnionym trójkątem
2. Definiuje rodzinę algorytmów, heremetyzuje każdy z nich i czyni je wymiennymi. Pozwala na dynamiczną zmianę algorytmu. Jaki to wzorzec?
 - A. Builder
 - B. **Strategy**
 - C. Facade
3. Dodaje funkcjonalność do klasy, nie zmieniając zachowania istniejących obiektów tej klasy. Pozwala na dodawanie zachowań w sposób dynamiczny. Jaki to wzorzec?
 - A. Strategy
 - B. **Decorator**
 - C. Simple Factory

Spis ilustracji

- <https://pixabay.com/images/id-2492009/> (wyzwanie)
- <https://pixabay.com/images/id-3303396/> (przykład)
- <https://www.pexels.com/pl-pl/zdjecie/czas-na-kapiel-guma-gumowa-kaczuszka-kaczatko-132464/> (kaczka gumowa)
- <https://www.pexels.com/pl-pl/zdjecie/ceramiczny-dekoracja-drewniany-drewno-1122562/> (kaczka drewniana)