Moduł 1

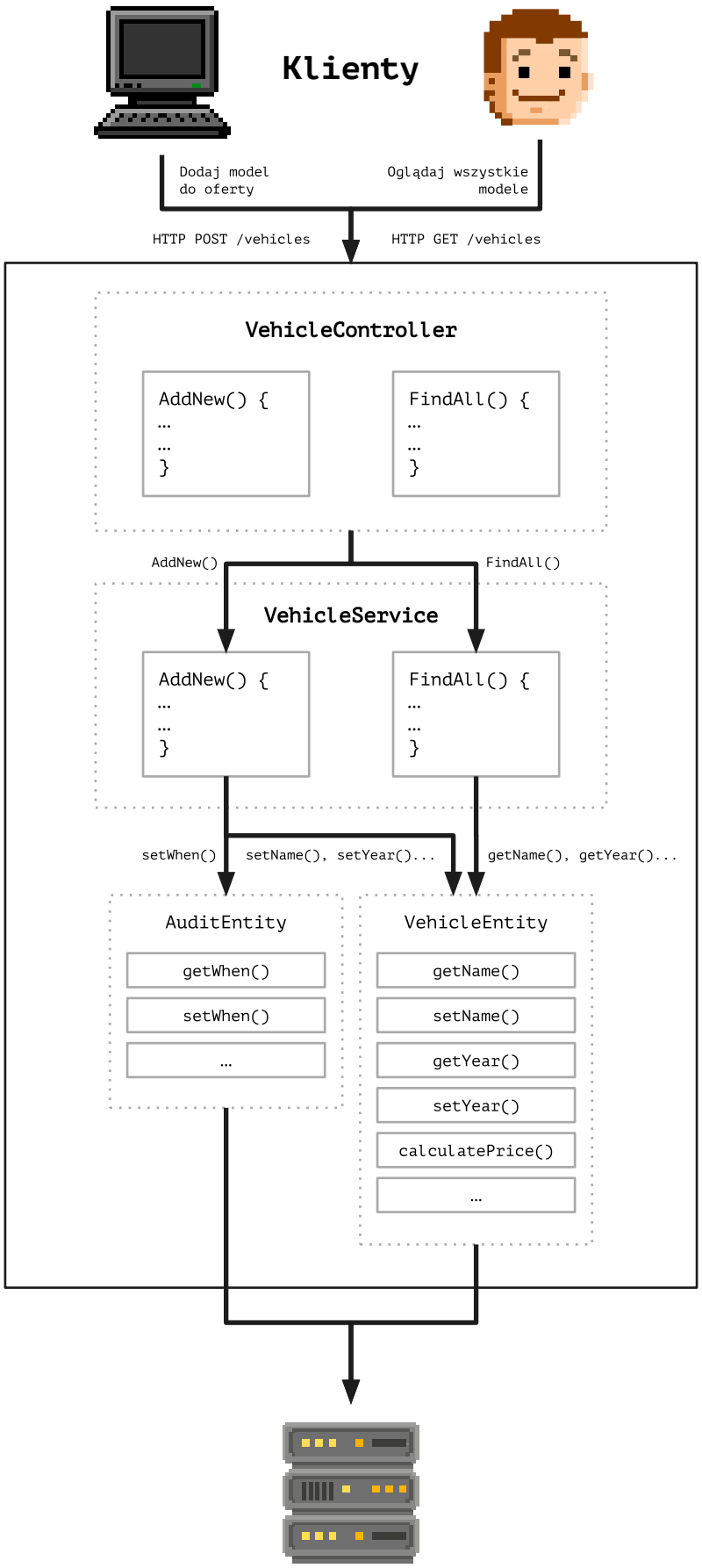
## Wprowadzenie

# Obserwowalne Zachowania i Wysokość Testów

LEGACY FIGHTER

# Obserwowalne Zachowania i Wysokość Testów

**Poniżej znajduje się uproszczony szkic architektoniczny systemu do sprzedawania samochodów.**



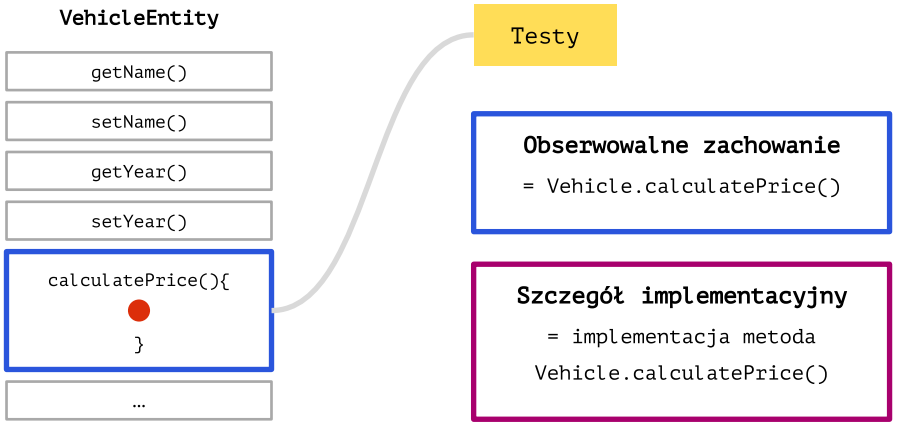
# Obserwowalne Zachowania i Wysokość Testów

Wyobraźmy sobie teraz kilka potencjalnych miejsc do refaktoryzacji. Załóżmy, że projekt nie posiada żadnych testów automatycznych. W zależności od refaktoryzowanego kawałka kodu, zobaczymy gdzie będzie można otoczyć się stabilnym interfejsem, który jest obserwowalnym zachowaniem. Na nim będziemy mogli wykonać test automatyczny.

Oto kilka typowych sytuacji związanych z refaktoryzacją:

## 1.

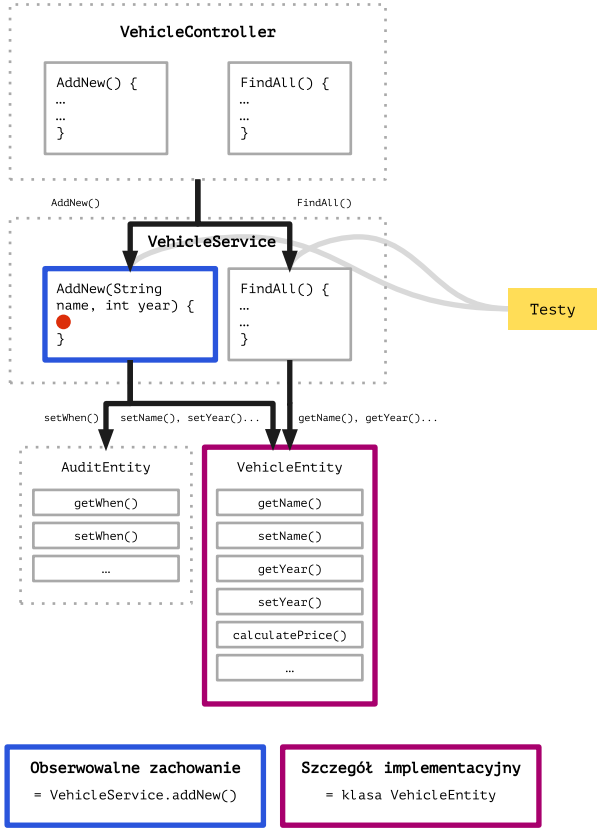
Należy dokonać poprawy funkcji liczącej cenę pojazdu. Miejsce do zmiany oznaczono czerwoną kropką. Sytuacja jest prosta - Encja Vehicle ma stabilny interfejs calculatePrice(), który jest obserwowalnym zachowaniem, które możemy testować.

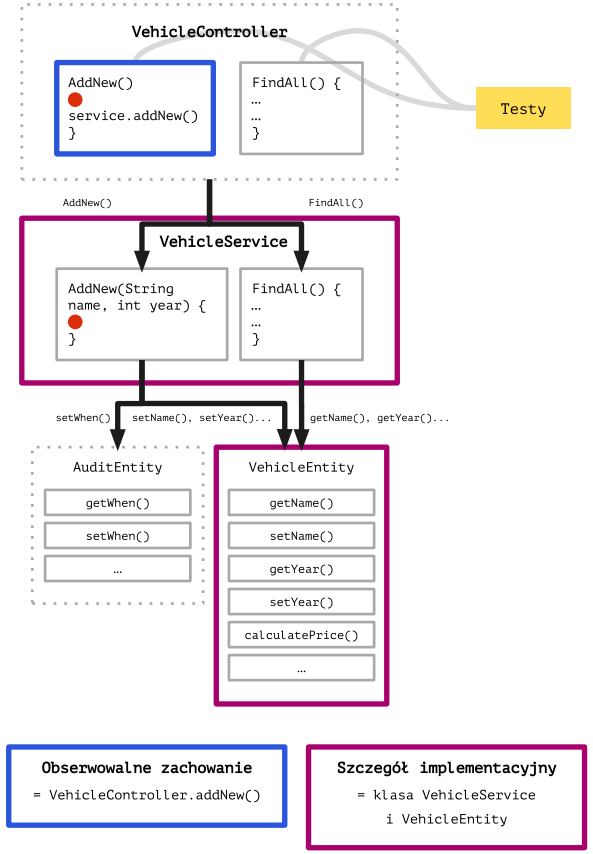


# Obserwowalne Zachowania i Wysokość Testów

## 2.

Należy przepisać walidacją nazwy nowo dodawanego modelu samochodu. Dodawanie odbywa się poprzez kontroler VehicleController i jego metodę addNew(). Metoda VehicleEntity.setName() nie ma żadnej logiki walidacyjnej. Praca odbywa się w serwisie, kontroler tylko deleguje pracę do wspominanego serwisu.



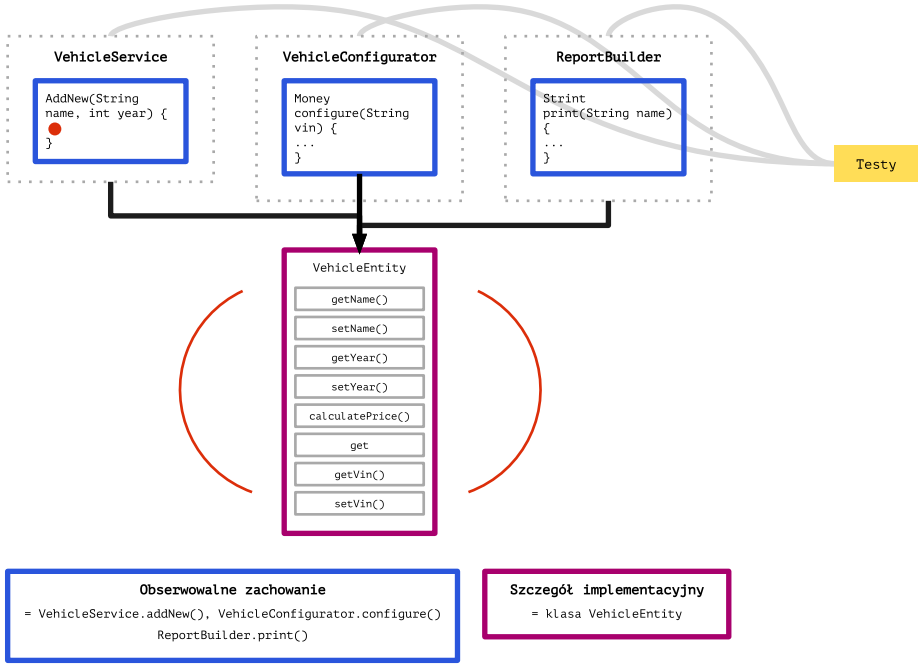
# Obserwowalne Zachowania i Wysokość Testów

W tym wypadku obserwowalnym zachowaniem jest metoda serwisu albo kontrolera. Do testu możemy wybrać metodę serwisu, gdyż kontroler nie posiada żadnej logiki - deleguję pracę do serwisu, więc nasz test nie pominie żadnego ważnego aspektu walidacji nazwy. Serwis jest stabilnym interfejsem ukrywającym logikę walidacji (rysunek po lewej stronie). Zauważmy, że klasa VehicleEntity, która w przypadku liczenia ceny zawierała stabilny interfejs z obserwowalnym zachowaniem, staje się teraz szczegółem implementacyjnym serwisu. Klienty serwisu o niej nie wiedzą. Jak w teście sprawdzić, że dodanie nowego modelu samochodu się udało? Warto sprawdzić czy metoda addNew zakończyła się sukcesem, ale dużo lepiej zadać sobie pytanie: na jakie inne obserwowalne zachowanie dodanie nowego modelu ma wpływ? Ma wpływ na pojawienie się tego modelu w metodzie findAll, której można użyć do testów.

Gdyby kontroler zawierał logikę walidacyjną, on stałby się miejscem do testu (obserwowalnym zachowaniem ze stabilnym interfrejsem - rysunek po prawej stronie). Klienty kontrolera nie wiedzą o serwisie, który teraz stał się szczegółem implementacyjnym. Nie możemy testować na wysokości serwisu, bo ominęlibyśmy część logika zawartej w kontrolerze. W niektórych przypadkach, jeśli taką logikę można byłoby bezpiecznie i mechanicznie przenieść do serwisu, obserwowalne zachowanie, a więc i stabilny interfejs do testu wróciłby do metody serwisowej. Zamiast testu end to end można w takich wypadkach zastosować test niższego poziomu, najpewniej integracyjny.

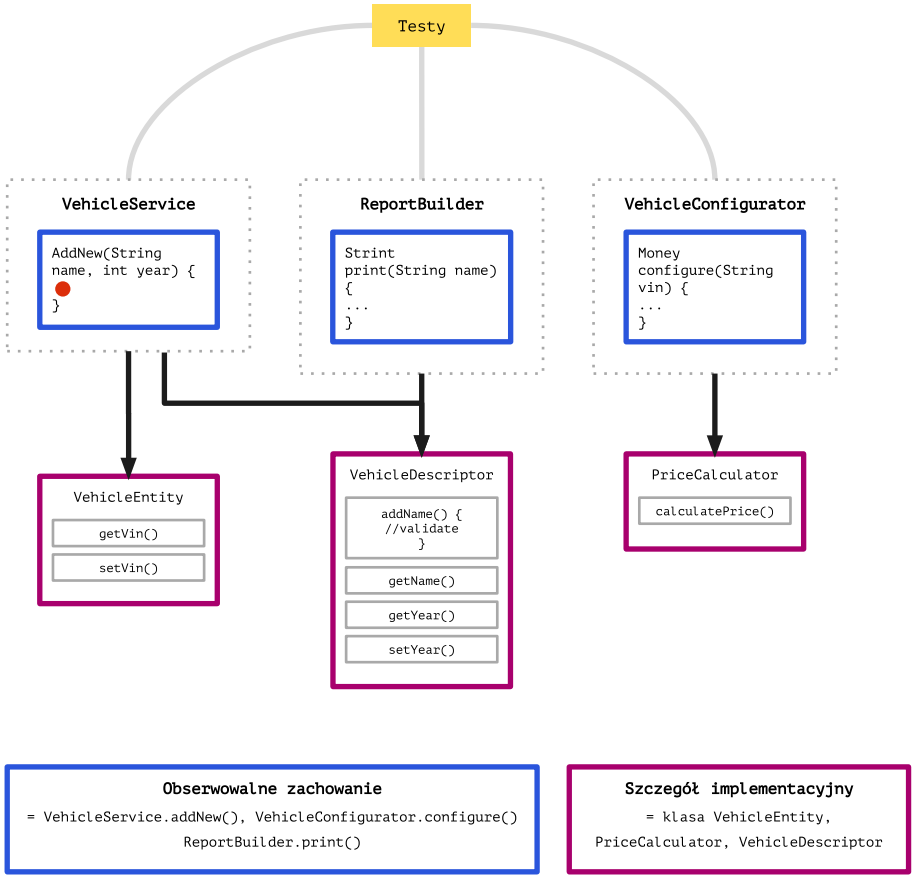
## 3.

Odkryto, że klasa VehicleEntity ma zbyt wiele odpowiedzialności i należy podzielić ją na trzy inne. Jeśli odpowiedzialności było dużom to z pewnością sporo jest klientów klasy VehicleEntity. Testy napisane na interfejsie VehicleEntity nie byłyby pomocne, ponieważ jej zakres odpowiedzialności się zmieni, więc pewnie zmieni się API, więc te testy też naturalnie uległyby zmianie. Klasa VehicleEntity jest tutaj tylko szczegółem implementacyjnym, na którym nie opierać testu. Stabilnego interfejsu trzeba szukać wyżej, na poziomie klientów klasy VehicleEntity.



# Obserwowalne Zachowania i Wysokość Testów

Dzięki temu, po wprowadzeniu zmiany, testy nie ulegną modyfikacji, więc mogą ciągle weryfikować czy nie wprowadzono żądnego błędu przy refaktoryzacji.




Oczywiście, jeśli któraś z nowo wprowadzonych klas ma teraz stabilny interfejs i reprezentuje jakieś obserwowalne zachowanie, możemy na tym infefejsie oprzeć testy. Przykładowo, testy sprawdzania poprawności nazwy modelu mogą teraz znaleźć się na wysokości klasy VehicleDescriptor, a nie na serwisie. Test niższego poziomu, np. jednostkowy, łatwiej będzie implementować. Wcześniej, nie było to możliwe, gdyż VehicleEntity operował tylko metodami dostępowymi typu get/set.