**Technologie obiektowe**

**Projekt**

**Temat:** Wzorce projektowe dotyczące mapowania obiektowo – relacyjnego.

**Autor:** Bartosz Dygas

**Prowadzący:** dr inż. Mariusz Bedla

Część teoretyczna

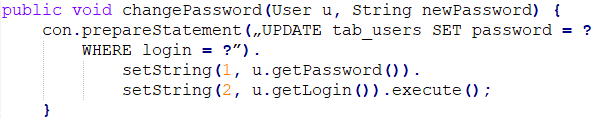
1. **Czym jest mapowanie obiektowo – relacyjne(ORM)?**

Mapowanie obiektowo – relacyjne można opisać jako sposób odwzorowania architektury obiektowego systemu informatycznego na bazę danych o charakterze relacyjnym.

Dokładniej – jest to konwertowanie tabel w bazie danych na obiekty aplikacji oraz na odwrót – konwertowanie obiektów aplikacji na tabele bazy danych.

ORM pozwala nam zastąpić poniższy kod wersją, w której nie ma konieczności bezpośrednich zmian w bazie danych z poziomu kodu.

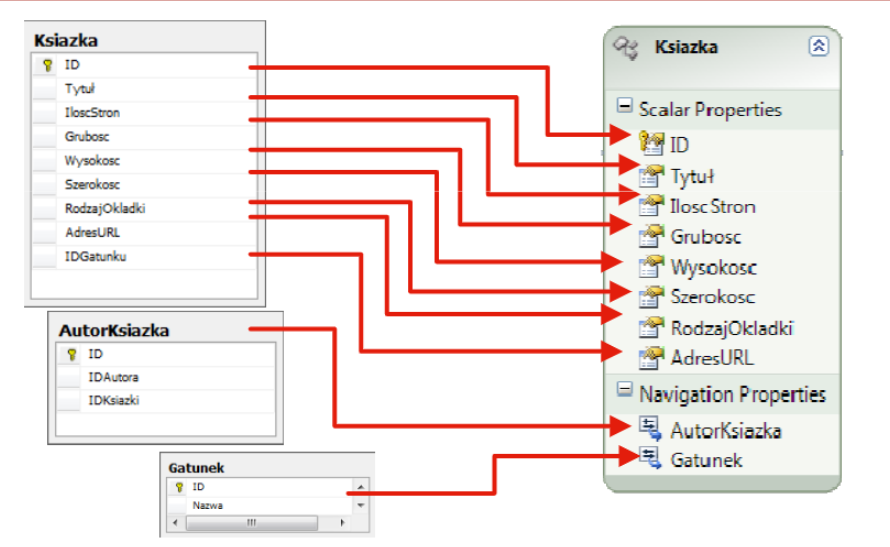
Wersja przed zastosowaniem ORM:



Wersja po zastosowaniu ORM:



ORM przedstawione graficznie:

[[1]](#footnote-1)

1. **Dlaczego stosować mapowanie obiektowo – relacyjne?**
2. **Uniezależnienie od rodzaju systemu zarządzania bazą danych(DBMS) oraz języka zapytań bazy danych.**

Dzięki wykorzystaniu odpowiedniej technologii (np. hibernate) można używać ORM bez znajomości języka SQL oraz systemu DBMS.

1. **Automatyzacja**

Dzięki wykorzystaniu ORM można zaprogramować aplikację tak, aby atrybuty obiektów automatycznie były zapisywane w bazie danych i na odwrót – tabele z baz danych były konwertowane i zapisywane do atrybutów obiektów.

1. **Redukcja ilości czasu potrzebnego na przepisywanie tabel do aplikacji lub obiektów aplikacji do tabel.**

Dzięki wykorzystaniu ORM nie tylko zmniejszamy objętość kodu aplikacji, ale też redukujemy czas potrzebny na przepisanie danych z lub do bazy danych. ORM manager zarządza optymalizacjami wydajnościowymi (pulą połączeń do bazy itp.).

1. **Uniknięcie niezgodności impedancji.**

Różnice w koncepcji języków powodują trudności techniczne w zrealizowaniu połączenia między językiem zapytań, a językiem programowania. Niezgodność impedancji to niezgodność m.in. składni, systemów typów, semantyki, poziomu abstrakcji języków. Dzięki wykorzystaniu ORM unikamy niezgodności impedancji.

1. **Jak działa mapowanie obiektowo – relacyjne?**

Mapowanie obiektowo– relacyjne polega na zdefiniowaniu odwzorowania zawartości relacyjnej bazy danych na obiekty w używanym przez nas języku programowania oraz wykonywaniu operacji na danych w bazie danych tak jak na zwykłych obiektach języka programowania.

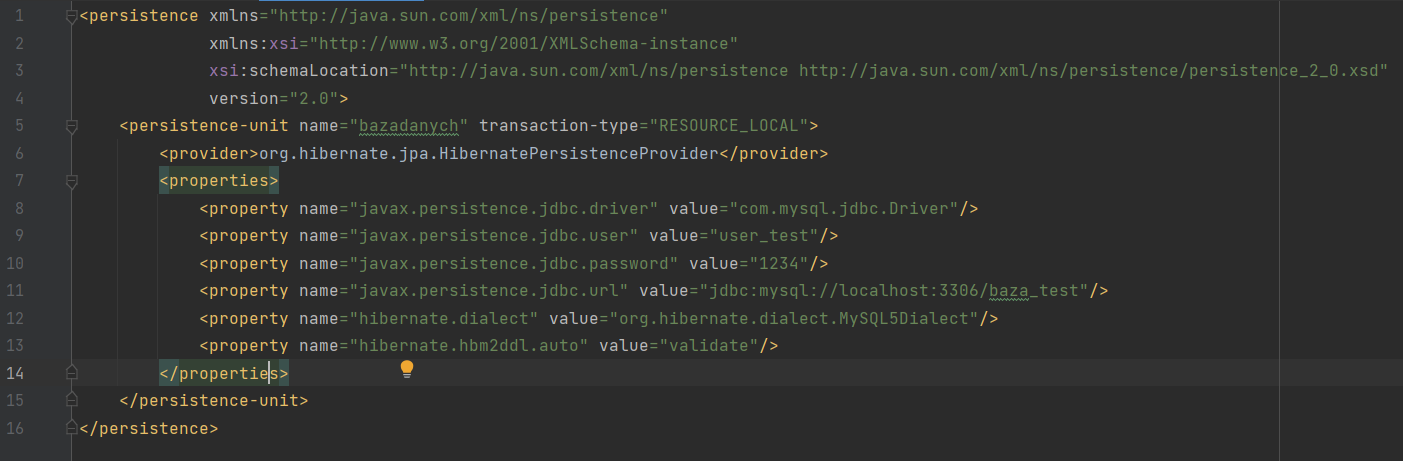
Zasady pracy z narzędziami ORM:

1. Tworzenie modelu danych w obiektowym języku programowania
2. Tworzenie schematu bazy danych odpowiadającego modelowi obiektowego języka (jeśli taka baza danych jeszcze nie istnieje)
3. Definiowanie odwzorowania bazy danych na model relacyjny
4. Tworzenie aplikacji w oparciu o zaprojektowany wcześniej model danych.
5. W razie konieczności pobrania obiektów z bazy danych, utrwalenia nowo utworzonego obiektu lub usunięcia utrwalonego obiektu – posługujemy się odpowiednim API danego narzędzia ORM. To jest jedyne miejsce, w którym w naszej aplikacji przejmujemy się tym, że współpracujemy z jakąś bazą danych.
6. **Technologie**

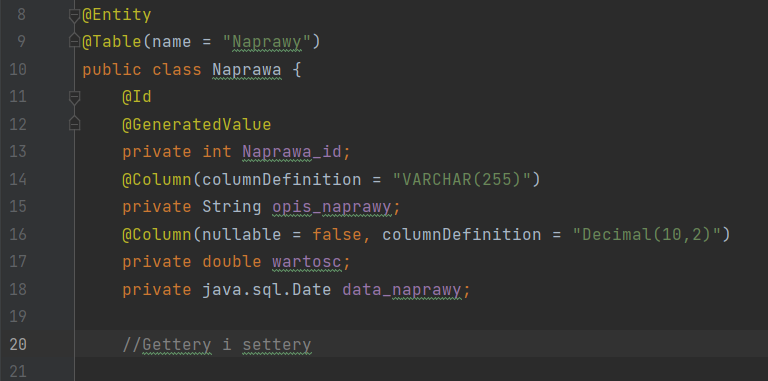
Najpopularniejsze technologie wykorzystywane w mapowaniu obiektowo – relacyjnym:

1. Java EE – platforma do tworzenia aplikacji biznesowych w języku Java posiadająca technologię ORM bazującą w dużej części na Hibernate. Początkowo wykorzystywała mało dziś popularny standard EJB, który dopiero w wersji 5 został zmieniony na standard JPA.
2. Spring – platforma dedykowana do tworzenia aplikacji w języku Java. Wykorzystuje framework hibernate i został stworzony jako konkurencja dla technologii Java EE.
3. **Hibernate** – Framework - technologia wykorzystywana podczas tworzenia ORM w języku Java. Jego dużą zaletą jest wykorzystywany przez niego standard JPA (Java Persistence API), czyli model mapowania obiektowo-relacyjnego, który zapewnia kompatybilność zarówno z bazami SQL jak i noSQL.

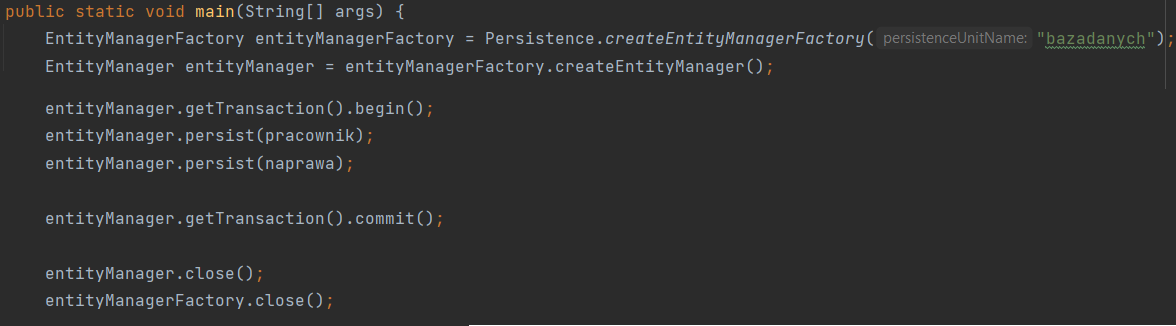
Głównym zadaniem Hibernate jest mapowanie klas języka Java na tabele bazy danych, w tym także mapowanie typów CLR (Common Language Runtime) na typy bazy danych. Jednocześnie oprogramowanie to dostarcza metod, służących do operowania na danych – zapisu, odczytu, edycji, usuwania. Hibernate jest bardzo popularnym rozwiązaniem ze względu na najbardziej rozbudowane API. Jednak przez nadmierne dodawanie funkcjonalności dokumentacja rozwiązań jest niepełna. Z tego powodu nauka tej technologii jest czasochłonna. Do poprawnej pracy wymagane jest utworzenie mapowań wskazujących odpowiednio, w jaki sposób obiekty mają być przechowywane w relacyjnej bazie danych.



Plik persistence.xml pozwalający na połączenie między aplikacją, a bazą danych przy wykorzystaniu Hibernate. W tym miejscu trzeba ustalić nazwę i lokalizację bazy danych, login, hasło użytkownika, typ operacji(create, validate, update itp.)



Klasa wykorzystująca Hibernate do mapowania obiektowo – relacyjnego.

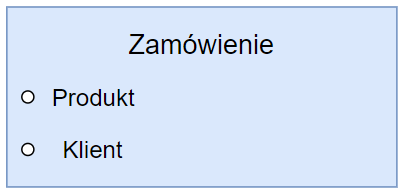


Wykorzystanie entity managera do dokonania zmian w bazie danych.

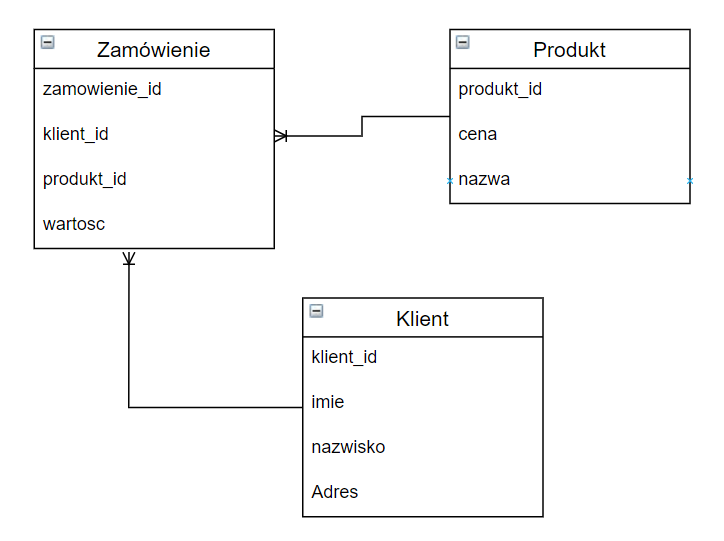
1. **Wzorce mapowania obiektowo – relacyjnego.**
2. **Wzorzec Repository**

Ściśle łączy się z ideą Domain Driven Design, która to z kolei zakłada, że nacisk powinien być położony na zagadnienia biznesowe, a nie na technologię. Pozwala to na modelowanie systemów informatycznych przez ekspertów, którzy znają specyfikę problemu, ale nie mają doświadczenia w projektowaniu architektury systemów informatycznych. W ramach DDD wyróżnia się „Domain objects”. Przykładowo – podczas realizacji zamówienia obiektem typu „Domain object” będzie zamówienie, które zawiera informację o zamawianym produkcie i kliencie. Natomiast na poziomie bazy danych są to 3 różne obiekty: klient, produkt, zamówienie.

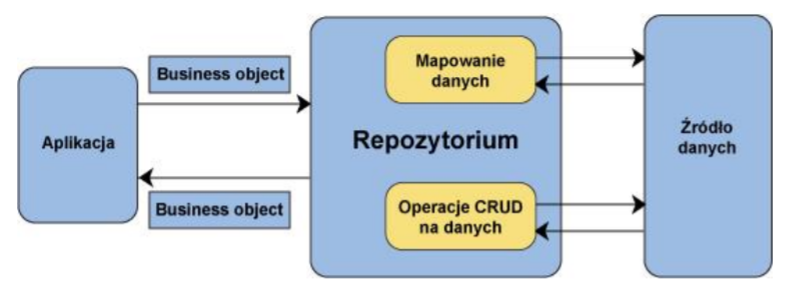
Domain object „Zamówienie”:



Obiekty w bazie danych:



Do obsługi obiektów Domain objects służa specjalne repozytoria. Dzięki nim, programiści mogą korzystać z danych zamodelowanych w taki sposób, aby odpowiadały zagadnieniom biznesowym bez konieczności tworzenia dodatkowych struktur danych. Repozytorium pośredniczy między aplikacją a warstwą mapowania danych.

[[2]](#footnote-2)

Business object na rysunku to synonim dla Domain object.

Dzięki użyciu repozytorium możliwe jest również połączenie się z kilkoma źródłami danych.

Zasady wzorca Repository:

1. **Ograniczony kontekst**

Wyraźnie ustalone granice pod względem organizacji zespołu, użycia w określonych częściach aplikacji i fizycznych manifestacji, takich jak bazy kodu i schematy baz danych

1. **Ciągła integracja**

Rozbicie systemu na małe konteksty powoduje utratę spójności i integracji. W tym celu należy wdrożyć proces częstego scalania całego kodu oraz wypracować wspólny pomysł na model, organizować spotkania wśród developerów, na których mogą opowiedzieć jaki jest ich pomysł na kontynuację projektu.

1. **Mapa kontekstowa**

Ograniczony kontekst sprawia, że powstaje brak ogólnego widoku projektu. Aby rozwiązać ten problem należy zdefiniować każdy model w projekcie.

**ADO.Net Entity Framework.**

Jest to implementacja wzorca repository oraz Domain Driven Design.  
Głównym założeniem jest istnienie trzech modeli danych: fizycznego, logicznego oraz koncepcyjnego. Model fizyczny odnosi się do specyficznej platformy przechowywania informacji w pamięci trwałej np. sposobu zapisu danych na dyskach twardych przez MS SQL. Najważniejsze dla programistów są jednak elementy Entity Framework:

* model koncepcyjny omawiający encje i relacje w ramach systemu, który jest analizowany
* model logiczny, czyli model koncepcyjny znormalizowany dla relacyjnych baz danych, opisany tabelami i relacjami między nimi.

W procesie tworzenia oprogramowania często pomija się model koncepcyjny, który jest stosowany podczas zbierania wymagań i opisu zależności w systemach. Zespoły programistyczne od razu przystępują do modelowania zagadnienia w sposób relacyjny. Entity Framework zakłada, że model koncepcyjny jest najbardziej zbliżony do rzeczywistych wymagań stawianych wobec systemów informatycznych. Z tego powodu został wprowadzony tzw. Entity Data Model (EDM), który jest specyfikacją mapowania pomiędzy modelem koncepcyjnym a modelem logicznym

1. **Unit of Work (Jednostka pracy)**

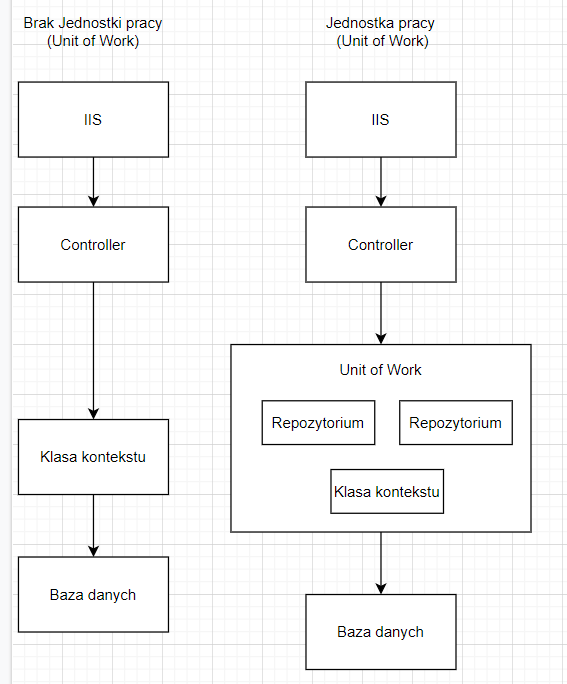
Jednostka pracy ma za zadanie śledzić wszystkie zmiany dokonywane podczas trwania transakcji. Sprawdza ona, czy występują jakiekolwiek niespójności oraz które obiekty w bazie danych zostały zmienione oraz dba o odpowiednią kolejność aktualizacji tak, aby więzy integralności nie zostały naruszone.  
Zadaniem jednostki pracy jest otworzenie transakcji, sprawdzenie współbieżności i zapisanie zmian w bazie danych. W ten sposób programista ma pewność, że integracja danych została zachowana.

W celu zachowania poprawnego działania jednostki pracy musi ona wiedzieć jakie obiekty musi śledzić. W tym celu można wykorzystać „Caller registration” lub „Object registration”

Podczas wykorzystania „Caller registration” użytkownik musi zarejestrować obiekt w jednostce pracy. Inaczej nie zostanie on zapisany.

Podczas wykorzystania „Object registration” metody rejestracji są umieszczone w metodach obiektu.

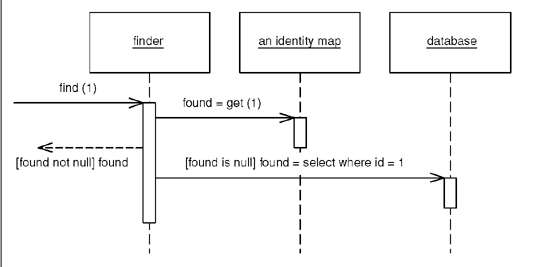
Wzorzec ten jest przydatny podczas dokonywania licznych zmian w bazie danych w krótkim czasie. Unit of Work pomaga zapewnić tym zmianom odpowiednią kolejność aktualizacji, przy czym śledzi je wszystkie i dba o zachowanie więzów integralności i o spójność danych.



1. **Identity Map (Mapa tożsamości)**

Mapa tożsamości zapewnia, że każdy obiekt zostanie załadowany do mapy tylko raz. W razie potrzeby jest on wyszukiwany na mapie.  
W ten sposób wzorzec ten rozwiązuje problem integralności i spójności danych, który może być wywołany załadowaniem jednego obiektu z bazy danych do więcej niż jednego obiektu programu. Dzięki temu poprawia się także szybkość działania aplikacji.   
Jedna mapa może przypadać na jedną tabelę w bazie danych. Za każdym razem gdy chcemy ładować obiekt z bazy danych – najpierw szukamy czy nie mamy go już w naszej mapie. Jeśli tak, to ładujemy obiekt z mapy. Jeśli nie, to umieszczamy obiekt z bazy danych na mapie.  
Problem pojawia się w momencie dziedziczenia. Można wykorzystywać różne mapy dla każdej klasy dziedziczącej lub jednej mapy dla całego drzewa dziedziczenia. Drugi sposób jest wygodniejszy lecz wymaga oprzeć mapy nie na tabelach bazy danych, ale na obiektach aplikacji.

Mapa tożsamości jest uniwersalnym rozwiązaniem i znajduje zastosowanie w wielu projektach. Należy szczególnie brać ją pod uwagę, gdy zależy nam, aby nie było możliwości w naszej aplikacji aby 2 obiektu (lub więcej) odpowiadały jednemu rekordowi bazy danych.  
Zaletą tego wzorca jest również to, że możemy uniknąć odwoływania się do bazy danych za każdym razem gdy potrzebujemy z niej jakichś informacji, ponieważ możemy mieć je już załadowane na mapie tożsamości. Pozwala to oszczędzić czas i przyspieszyć działanie aplikacji.

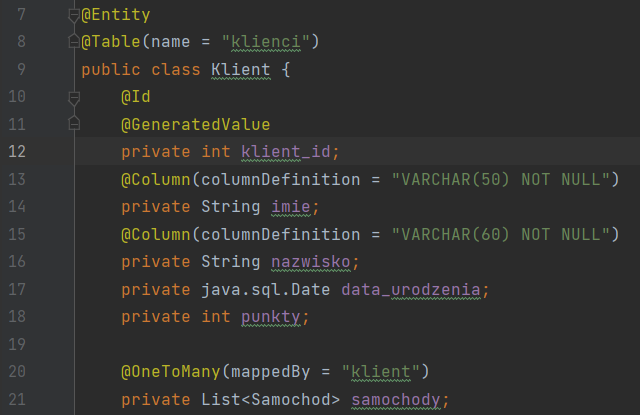
[[3]](#footnote-3)

1. **Foreign Key Mapping (Odwzorowanie do klucza obcego)**

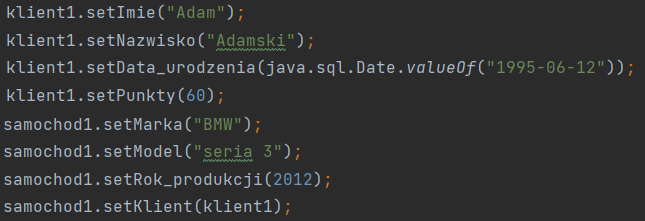
Mapowanie klucza obcego mapuje odwołanie do obiektu na klucz obcy w bazie danych. Połączenie między dwoma obiektami jest reprezentowanie przez klucz obcy znajdujący się w bazie danych. W ten sposób, widząc jeden obiekt, możemy łatwo ustalić inne obiekty, które są z nim połączone owym kluczem.

Przykład: klient i należący do niego samochód jako klasy:



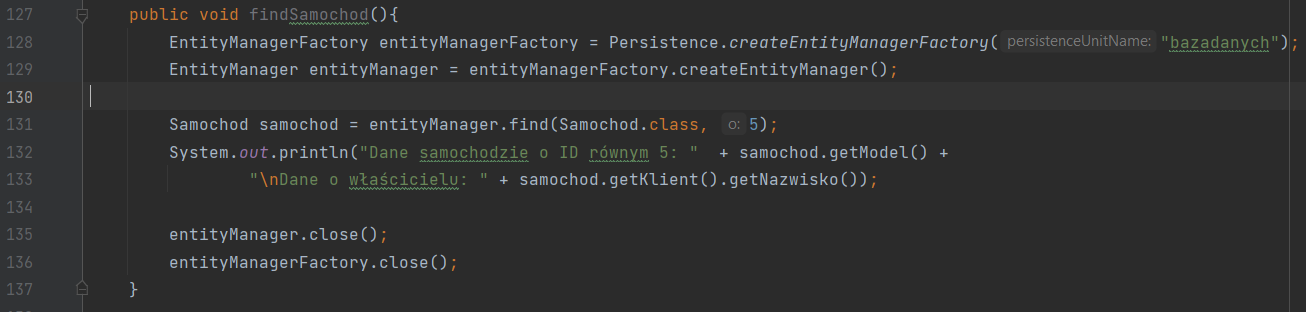


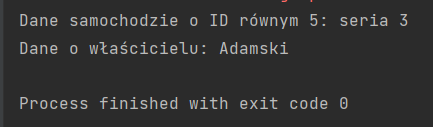
Ustawienie danych klienta i samochodu:



Klient w stosunku do samochodu posiada relację jeden do wielu. Jeden klient może mieć wiele samochodów, ale jeden samochód może należeć tylko do jednego klienta.

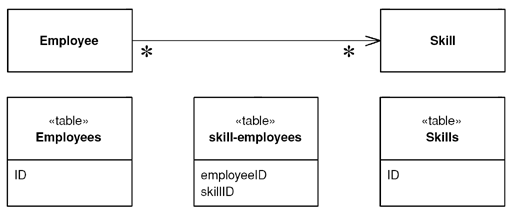
Wyszukiwanie właściciela konkretnego samochodu, pokazujące istnienie relacji między klientem, a samochodem:



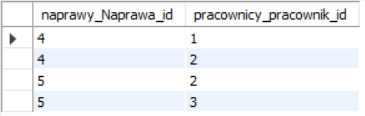


1. **Assosiation Table Mapping (odwzorowanie do tabeli asocjacji)**

Wzorzec ten sprawdza się w przypadku zastosowania relacji wiele do wielu. Posługując się nim, tworzymy nową tabelę zawierającą zarejestrowane relacje. Tabela ta zawiera wyłącznie identyfikatory kluczy obcych dla dwóch, połączonych tabel. Primary Key tabeli jest kluczem złożonym z dwóch wierszy – dwóch kluczy skojarzonych ze sobą tabel.   
Dlatego – w celu załadowania danych z tabeli zawierającej takie relację wymagane jest użycie dwóch zapytań.  
Wzorzec ten nie jest polecany dla asocjacji innych niż wiele do wielu, ponieważ jest on kosztowny i jego użycie może okazać się czasochłonne.

[[4]](#footnote-4)

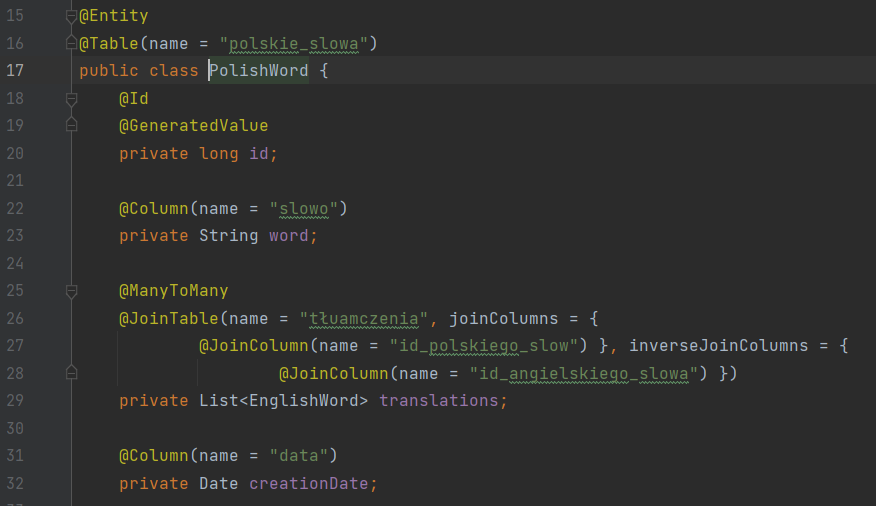
W poniższym przykładzie wykorzystałem 2 klasy zawierające relacje wiele do wielu. Dzięki właściwościom technologii Hibernate, przy odpowiednim mapowaniu tabele asocjacji tworzą się w tym przypadku automatycznie.

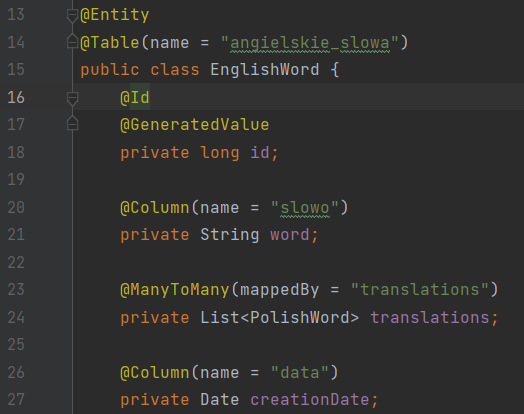


1. **Zastosowanie mapowania obiektowo-relacyjnego w praktyce**

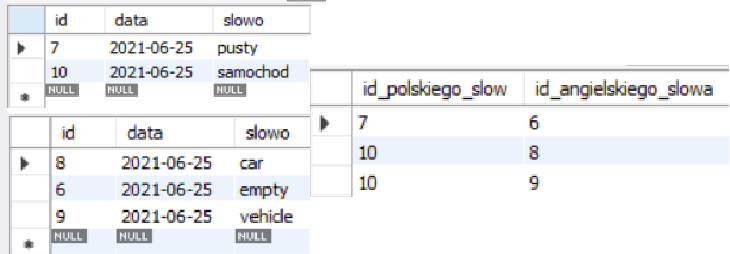
W ramach części praktycznej zastosowania ORM zaprojektowałem aplikację z fiszkami do nauki języka angielskiego. Aplikacja ta mapuje do bazy danych 2 obiekty – „PolishWord” i „EnglishWord”, w stosunku do których istnieje relacja wiele do wielu, ponieważ jedno słowo polskie może mieć kilka odpowiedników angielskich i na odwrót – jedno słowo angielskie może mieć wiele odpowiedników polskich. W takim wypadku zastosowałem tutaj omawiany wyżej wzorzec – odwzorowanie do tabeli asocjacji. Czyli w bazie danych, oprócz utworzenia tabel dla słów polskich i angielskich, tworzona jest także tabela „tłumaczenia” która zawiera identyfikatory kluczy wspomnianych dwóch tabel.

Klasy „PolishWord” i „EnglishWord”:



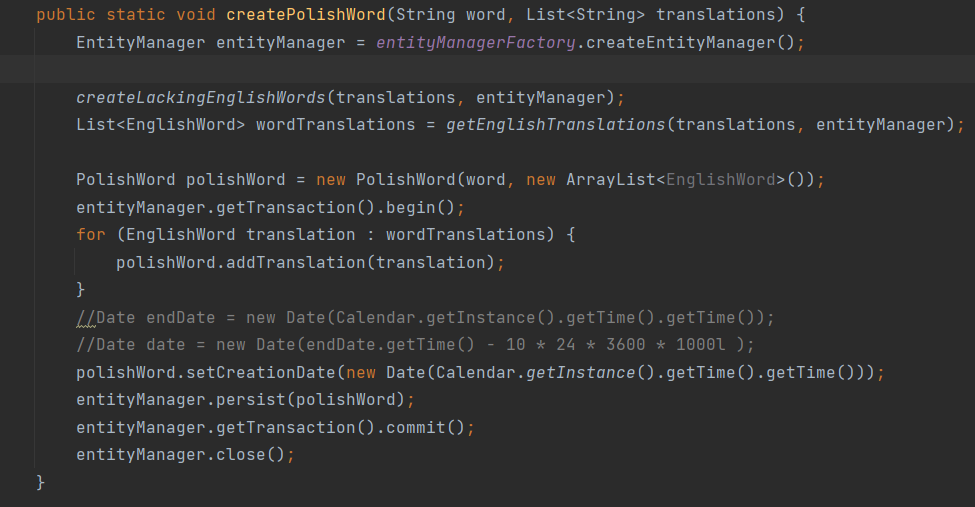


Odwzorowanie tabel w bazie danych:

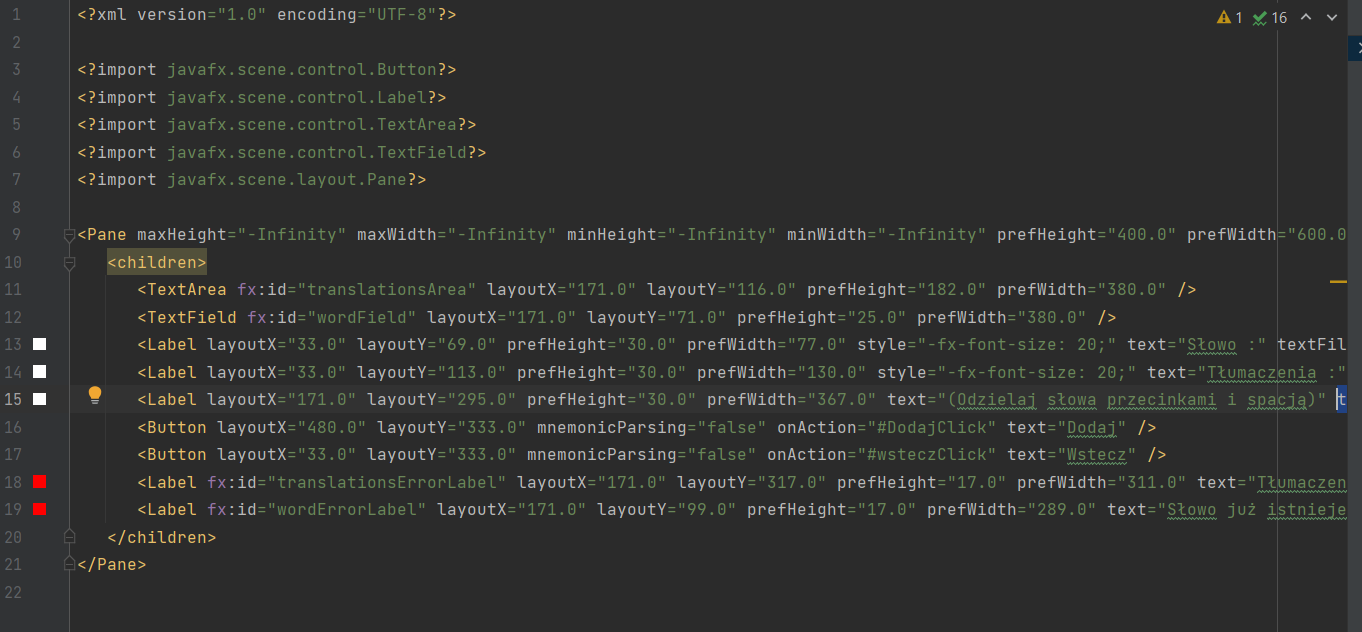


Aplikacja zaprojektowana jest zgodnie ze wzorcem Model-View-Controller i część graficzna zrobiona została przy zastosowaniu platformy JavaFX.

Kontrolery odpowiada za zmianę widoków, a modele zawierają klasy słów polskich, angielskich oraz zapisują słowa w bazie danych przy zastosowaniu technologii Hibernate.



Metoda zapisująca polskie słowo w bazie danych.



Klasa widoku w JavaFX.

1. <https://silo.tips/download/mapowanie-obiektowo-relacyjne-z-wykorzystaniem-hibernate> z dnia 15.04.2021r. [↑](#footnote-ref-1)
2. <http://users.pja.edu.pl/~mtrzaska/Files/PraceMagisterskie/091026-Niegowski.pdf> z dnia 22.04.2021r. [↑](#footnote-ref-2)
3. Martin Fowler, Architektura systemów zarządzania przedsiębiorstwem. Wzorce projektowe. S.178 [↑](#footnote-ref-3)
4. Martin Fowler, Architektura systemów zarządzania przedsiębiorstwem. Wzorce projektowe. S.226 [↑](#footnote-ref-4)