

# Hibernate, JPA – laboratorium

Instrukcja zawiera dwie części

Część I – mocno „przewodnikowa” – aby pomóc Państwu dojść do „czegoś działającego”.

Część II – (od punktu II włącznie) to co stanowi przedmiot Państwa (bardziej) samodzielnej pracy (w tym zadania domowego jeśli nie uda się skończyć w trakcie zajęć).

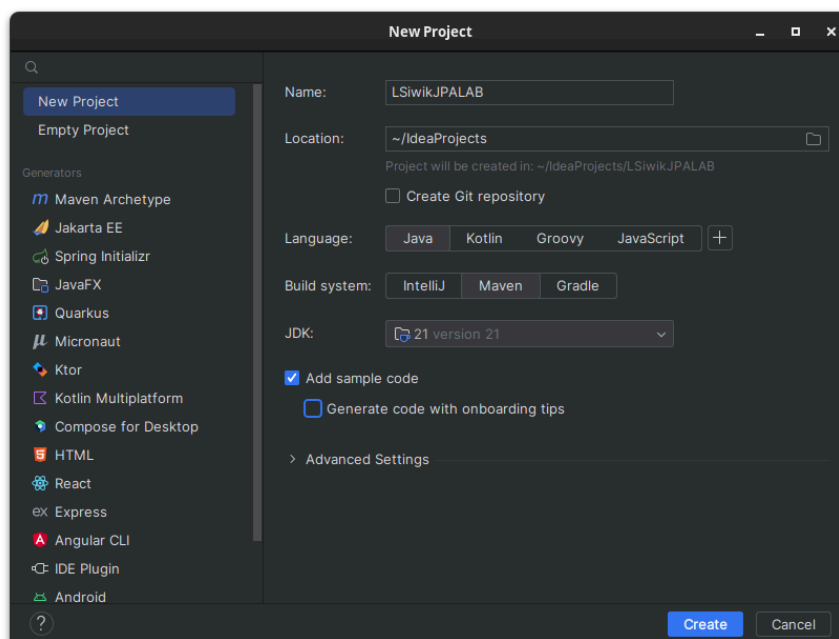
Na koniec zajęć proszę o umieszczenie w moodlu screenshot’a/krótkiego podsumowania pokazującego to co udało się zrealizować

## I. Basics

- Zacznijmy od ściągnięcia i rozpakowania serwera bazodanowego [Apache Derby](https://db.apache.org/derby/derby_downloads.html) w oparciu o który będziemy realizowali nasze ćwiczenie. Wędrujemy zatem do: [https://db.apache.org/derby/derby\\_downloads.html](https://db.apache.org/derby/derby_downloads.html) i pobieramy wersję serwera stosownie do wersji Javy którą mamy na komputerze na którym pracujemy)
- Uruchamiamy serwer Derby. Skrypt startowy (startNetworkServer) znajduje się w podkatalogu bin ściągniętej paczki. W przypadku powodzenia powinniśmy uzyskać efekt podobny do poniższego:

```
siwik@fedora:~/JPA-Lab/db-derby-10.16.1 (1).1-bin (1)/db-derby-10.16.1.1-bin/bin$  
./startNetworkServer  
Tue May 07 16:55:53 CEST 2024 : Apache Derby Network Server - 10.16.1.1 - (1901046  
) started and ready to accept connections on port 1527
```

- Zostawiamy sobie serwer pracującym i przechodzimy do tworzenia aplikacji.
- Wędrujemy do IntelliJ’a.
  - Tworzymy nowy projekt „Jawowo – Mavenovy”,



- ii. Po utworzeniu projektu (i ew. testowym uruchomieniu żeby sprawdzić czy na tym etapie wszystko jest ok) zaczniemy od dodania do projektu potrzebnych zależności.
- iii. Wędrujemy zatem do pom.xml i dodajemy zależności do:

1. Hibernate'a:

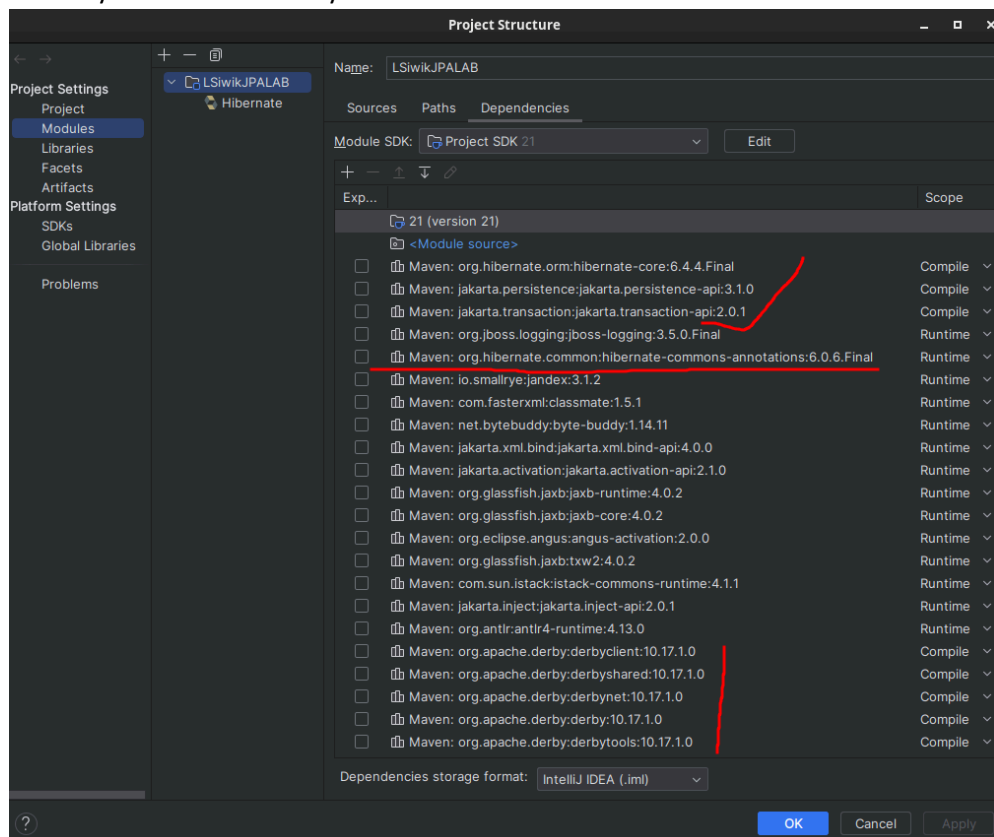
```
a. <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.4.4.Final</version>
</dependency>
```

2. Oraz do obsługi Apache-Derby:

```
a. <dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derbyclient</artifactId>
    <version>10.17.1.0</version>
</dependency>

b. <dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derbynet</artifactId>
    <version>10.17.1.0</version>
</dependency>
```

- e. Po zsynchronizowaniu mavena, powinniśmy w zależnościach projektu zobaczyć m.in to na czym nam zależało czyli Hibernate'a



- f. Możemy zatem zacząć pracować z Hibernate'm. Zaczniemy od dodania do projektu pliku pliku konfiguracyjnego (hibernate.cfg.xml), który przy standardowych ustawieniach projektu IntelliJ powinien znaleźć się w podkatalogu resources (zatem right click na resources -> new -> File -> hibernate.cfg.xml)
- g. I dodajemy do niego konfigurację potrzebną do nawiązania połączenia z uruchomionym serwerem Derby zmieniając nazwę bazy danych na ImieNazwiskoDatabase:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property
name="connection.url">jdbc:derby://127.0.0.1/MyLabDatabase;create=true</prope
rty>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hbm2ddl.auto">create-drop</property>
    </session-factory>
</hibernate-configuration>
```

- h. Uzupełnijmy klasę main na razie na tyle żeby spróbować otworzyć i zamknąć sesję do serwera / bazy danych

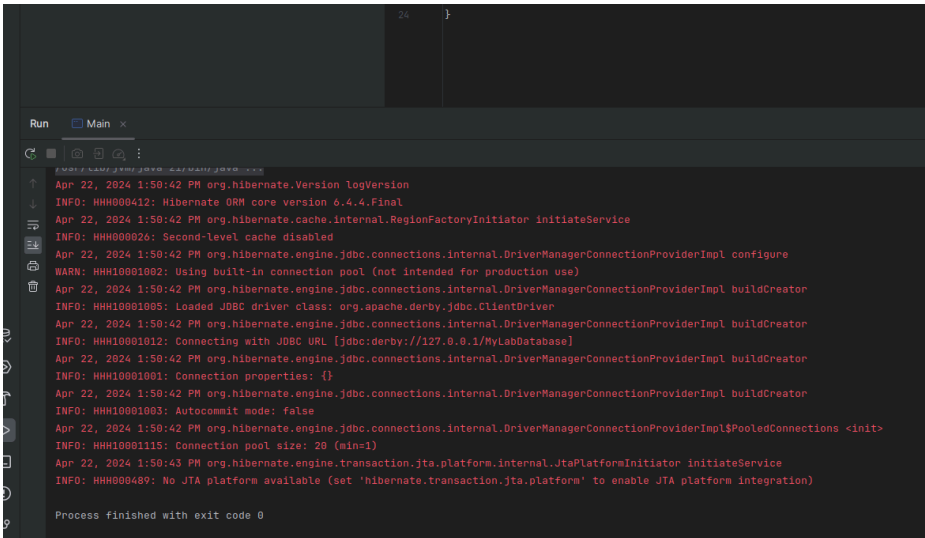
```
i. import org.hibernate.Session;
j. import org.hibernate.SessionFactory;
k. import org.hibernate.cfg.Configuration;
l.
m. public class Main {
n.     private static SessionFactory sessionFactory = null;
o.
p.     public static void main(String[] args) {
q.         sessionFactory = getSessionFactory();
r.         Session session = sessionFactory.openSession();
```

```

s.         session.close();
t.     }
u.
v.     private static SessionFactory getSessionFactory() {
w.         if (sessionFactory == null) {
x.             Configuration configuration = new Configuration();
y.             sessionFactory =
z.             configuration.configure().buildSessionFactory();
aa.         }
bb.         return sessionFactory;
cc.     }

```

- i. I spróbujemy uruchomić stworzonego main'a. Jeżeli wszystko poszło ok, na konsoli powinniśmy zobaczyć wpisy hibernate'a, bez żadnych wyjątków, czyli mniej więcej coś takiego jak poniżej:



```

24     }

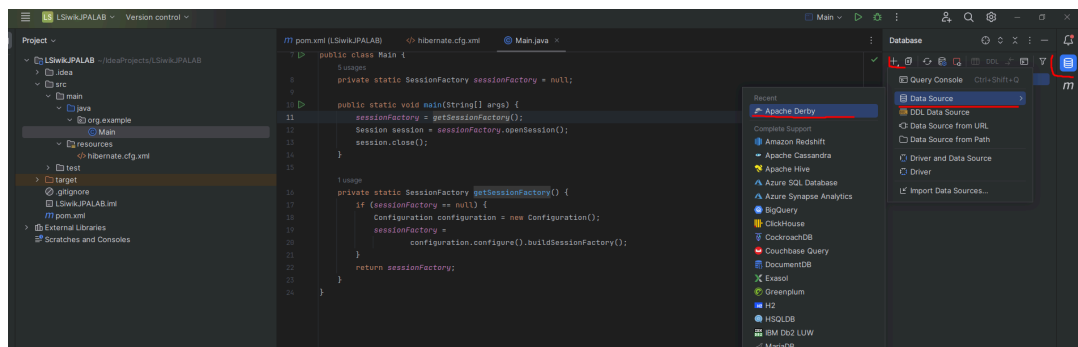
Run Main x
Apr 22, 2024 1:50:42 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate ORM core version 6.4.4.Final
Apr 22, 2024 1:50:42 PM org.hibernate.cache.internal.RegionFactoryInitiator initiateService
INFO: HHH000026: Second-level cache disabled
Apr 22, 2024 1:50:42 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using built-in connection pool (not intended for production use)
Apr 22, 2024 1:50:42 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: Loaded JDBC driver class: org.apache.derby.jdbc.ClientDriver
Apr 22, 2024 1:50:42 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001012: Connecting with JDBC URL [jdbc:derby://127.0.0.1/MyLabDatabase]
Apr 22, 2024 1:50:42 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {}
Apr 22, 2024 1:50:42 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
Apr 22, 2024 1:50:42 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH10001115: Connection pool size: 20 (min=1)
Apr 22, 2024 1:50:43 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)

Process finished with exit code 0

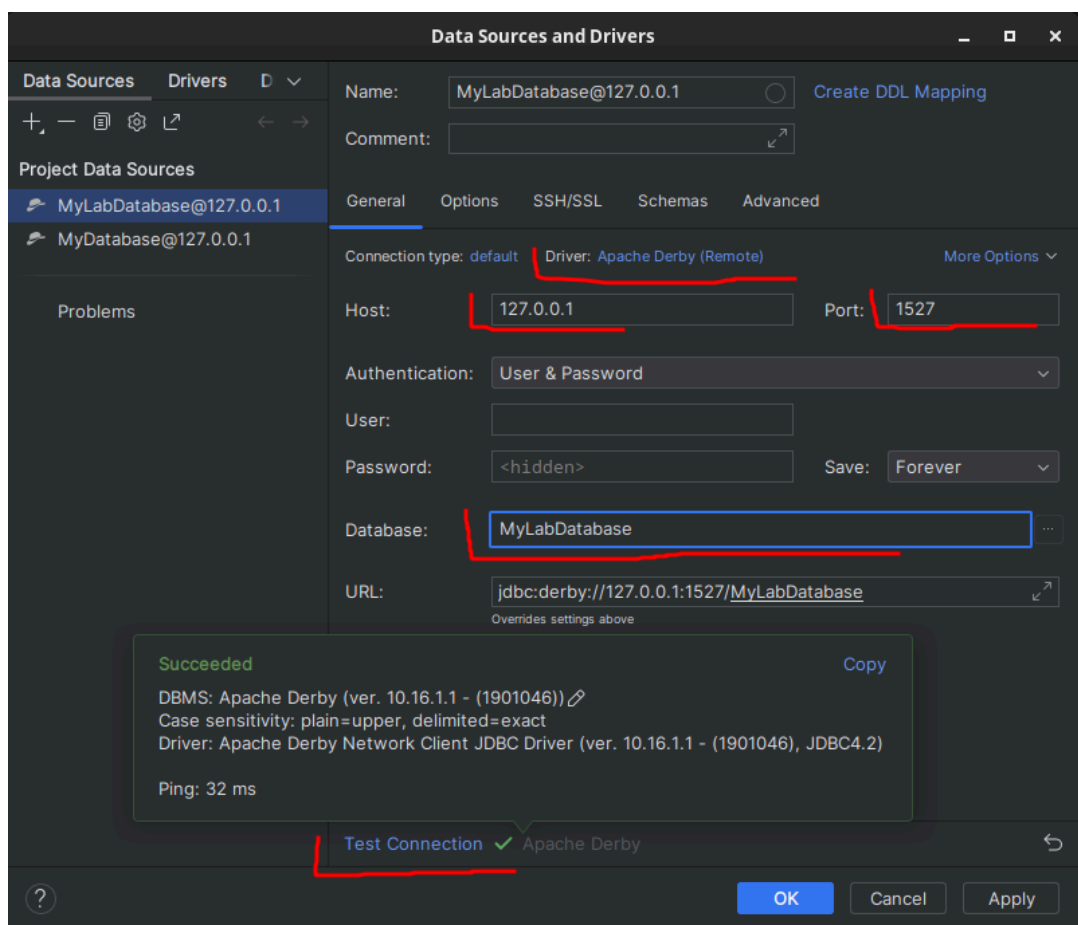
```

- j.
- k. Jeśli poprzednie się powiodło, na serwerze bazodanowym powinna się założyć baza o zdefiniowanej w hibernate config'u nazwie (w moim przypadku MyLabDatabase).
- l. Możemy to zweryfikować podpinając się np. z poziomu IntelliJ do uruchomionego serwera Derby do bazy o podanej nazwie i to połączenie powinno się powieść.

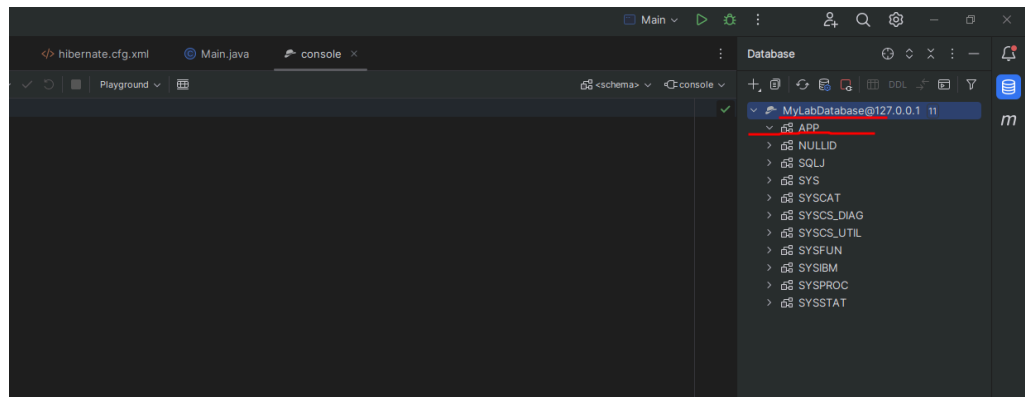
m.



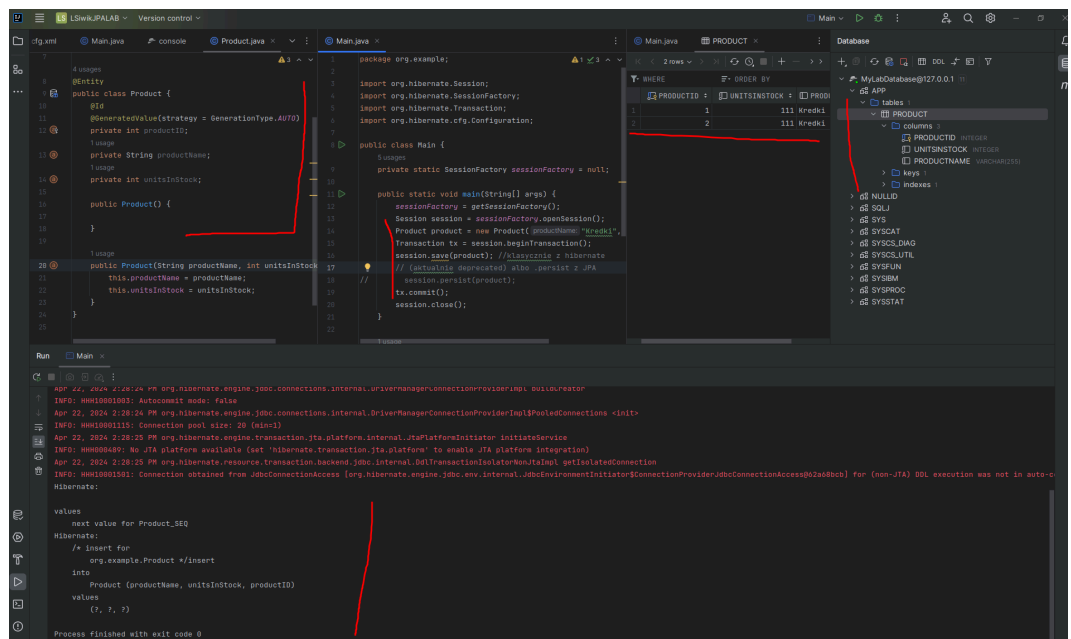
n.



- o. Po podpięciu do bazy na serwerze powinien być widoczny schemat APP – na razie pusty bo nie dodawaliśmy tam żadnych tabel – czyli stan jak poniżej

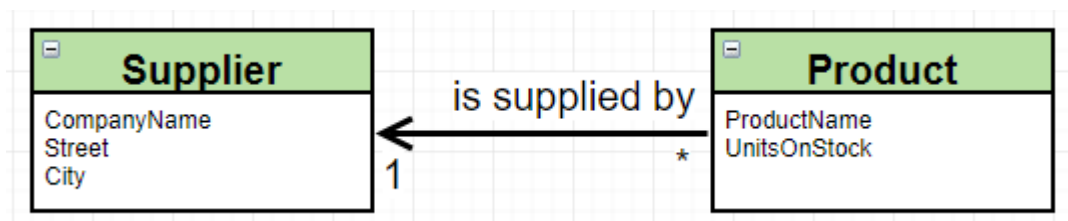


- p.
- q. No to stwórzmy coś w naszej bazie danych.
- r. Dodajmy do projektu klasę produktu z polami ProductName, UnitsOnStock
- s. Uzupełnijmy jej definicję o elementy niezbędne do jej zmapowania do bazy danych przez Hibernate (adnotacja @Entity, nominowanie pola ID, pusty konstruktor).
- t. Dodajemy w hibernate-config'u mapping dla stworzonej klasy.
- u. Rozszerzamy naszego maina o stworzenie nowego produktu i zapisanie go w bazie danych z wykorzystaniem hibernate'a. Uruchamiamy i testujemy projekt. Efekty powinny być takie/podobne jak poniżej:



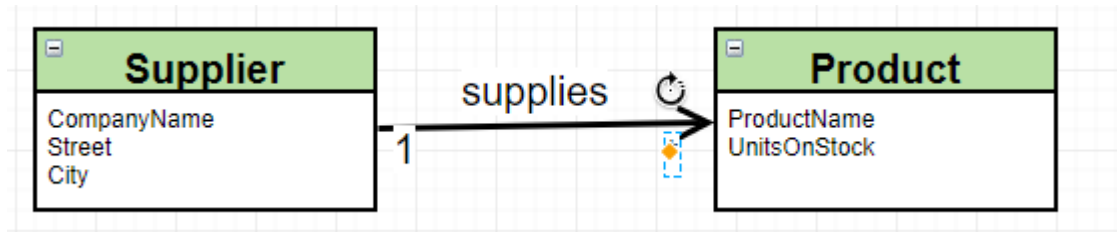
- v.
- w. Potwierdź w sprawozdaniu wykonanie powyższych kroków

- l. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej



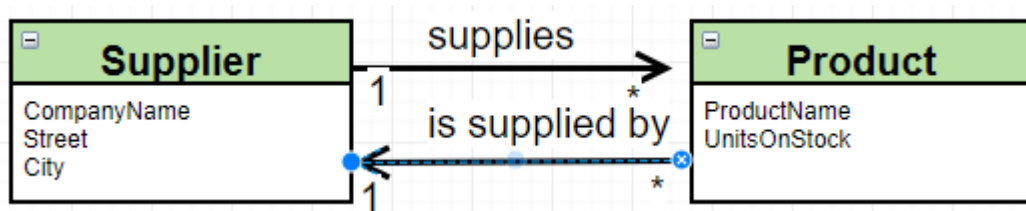
- a. Stworz nowego dostawce.
- b. Znajdz poprzednio wprowadzony produkt i ustaw jego dostawce na właśnie dodanego.
- c. **Udokumentuj wykonane kroki oraz uzyskany rezultat (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

II. Odwróć relacje zgodnie z poniższym schematem



- a. Zamodeluj powyższe w dwóch wariantach „z” i „bez” tabeli łącznikowej
- b. Stworz kilka produktow
- c. Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę
- d. **Udokumentuj wykonane kroki oraz uzyskane rezultaty w obu wariantach (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

III. Zamodeluj relację dwustronną jak poniżej:

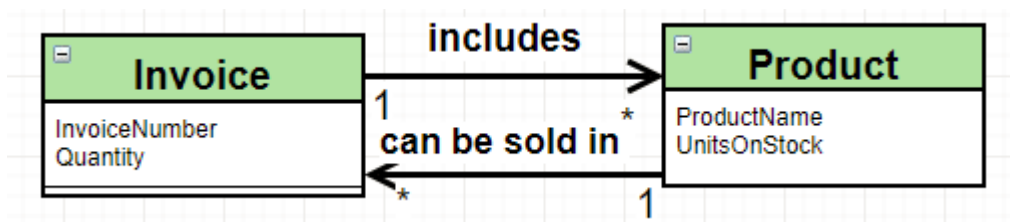


- a. Tradycyjnie: Stworz kilka produktow
- b. Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę (pamiętaj o poprawnej obsłudze dwustronności relacji)
- c. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

IV. Dodaj klase Category z property int CategoryID, String Name oraz listą produktow List<Product> Products

- a. Zmodyfikuj produkty dodając wskazanie na kategorie do której należy.
- b. Stworz kilka produktow i kilka kategorii
- c. Dodaj kilka produktów do wybranej kategorii
- d. Wydobądź produkty z wybranej kategorii oraz kategorię do której należy wybrany produkt
- e. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

V. Zamodeluj relacje wiele-do-wielu, jak poniżej:



- Stórz kilka produktów i “sprzedaj” je na kilku transakcjach.
- Pokaż produkty sprzedane w ramach wybranej faktury/transakcji
- Pokaż faktury w ramach których był sprzedany wybrany produkt
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

VI. JPA

- Stwórz nowego maina w którym zrobisz to samo co w poprzednim ale z wykorzystaniem JPA
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

VII. Kaskady

- Zmodyfikuj model w taki sposób aby było możliwe kaskadowe tworzenie faktur wraz z nowymi produktami, oraz produktów wraz z nową fakturą
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

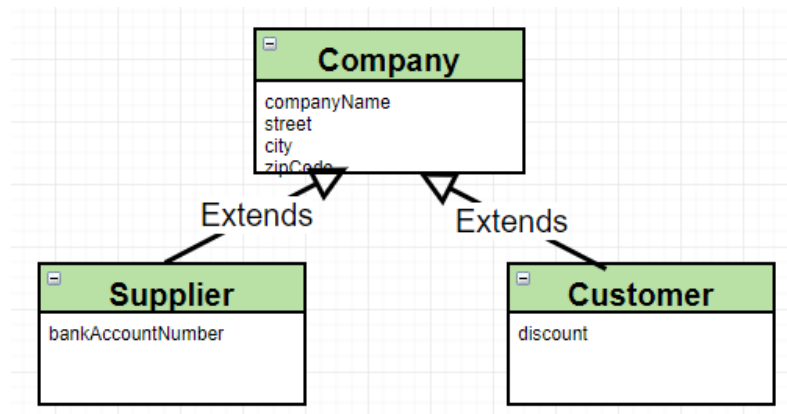
VIII. Embedded class

- Dodaj do modelu klasę adres. „Wbuduj” ją do tabeli Dostawców.
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat, select \* from....)**
- Zmodyfikuj model w taki sposób, że dane adresowe znajdują się w klasie dostawców. Zmapuj to do dwóch osobnych tabel.
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**

IX. Dziedziczenie



- a. Wprowadź do modelu następującą hierarchie:



- b. Dodaj i pobierz z bazy kilka firm obu rodzajów stosując po kolei trzy różne strategie mapowania dziedziczenia.
- c. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select \* from....)**