

# Hibernate

## zadania

1. Czynności wprowadzające.

Utworzyć bazę danych o nazwie: **sda\_hibernate**.

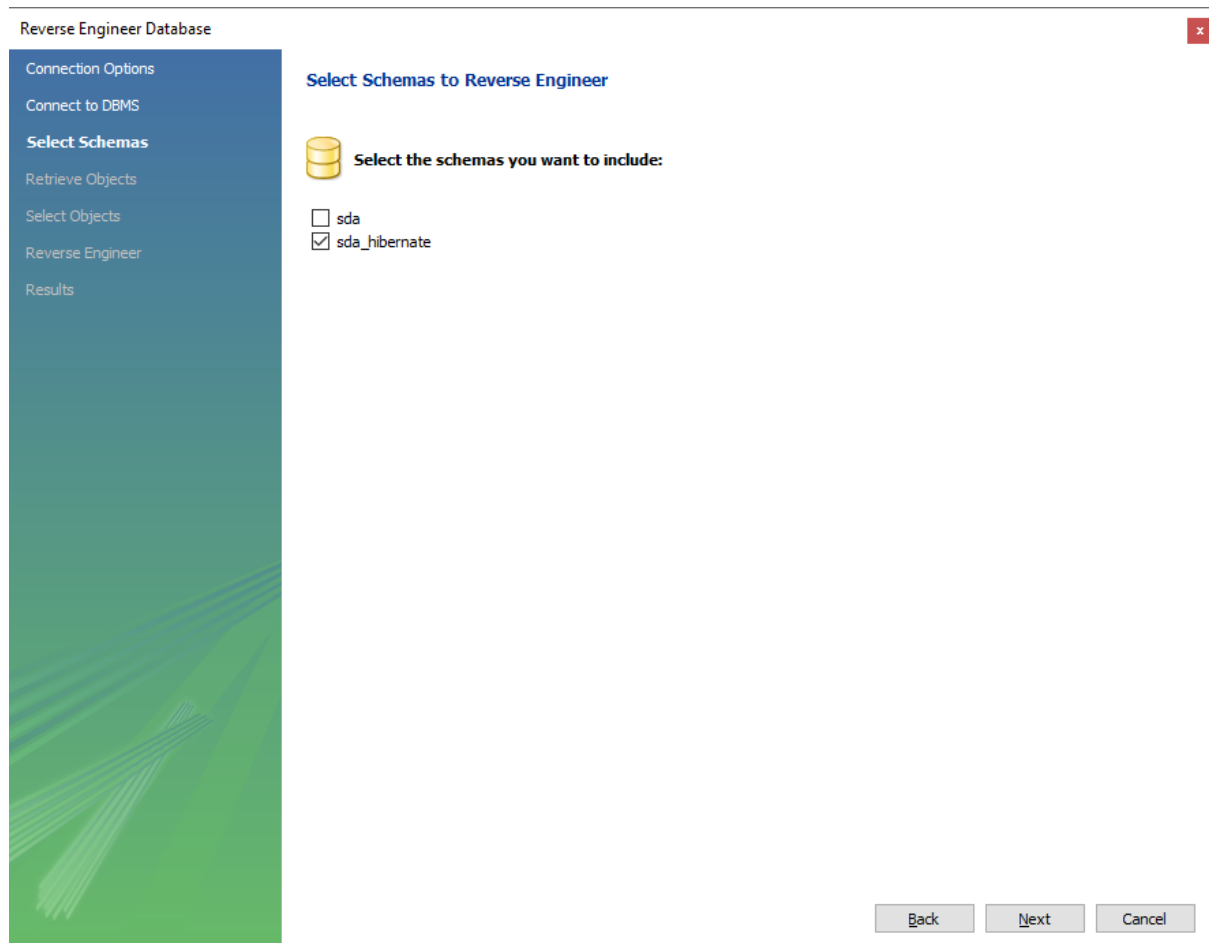
Następnie, w ramach nowo utworzonej bazy danych wykonać kolejno skrypty:

**skrypt\_init\_db\_20200808\_hibernate\_schemat.sql,**

**skrypt\_init\_db\_20200808\_hibernate\_data.sql.**

Następnie, należy zapoznać się ze schematem bazy danych przedstawionym na diagramie **schemat\_db.png**. Proszę zwrócić szczególną uwagę na istniejące relacje między tabelami.

Uwaga (ciekawostka). Po wykonaniu powyższych skryptów, diagram ten można uzyskać z poziomu *MySQLWorkbench*, dokonując „wstecznej inżynierii”. W tym celu, z menu kolejno wybieramy: Database / Reverse Engineer... (Ctrl + R). Następnie postępujemy zgodnie z procesem, wybierając na jednym z ekranów właściwą bazę danych, na podstawie której utworzony zostanie diagram.

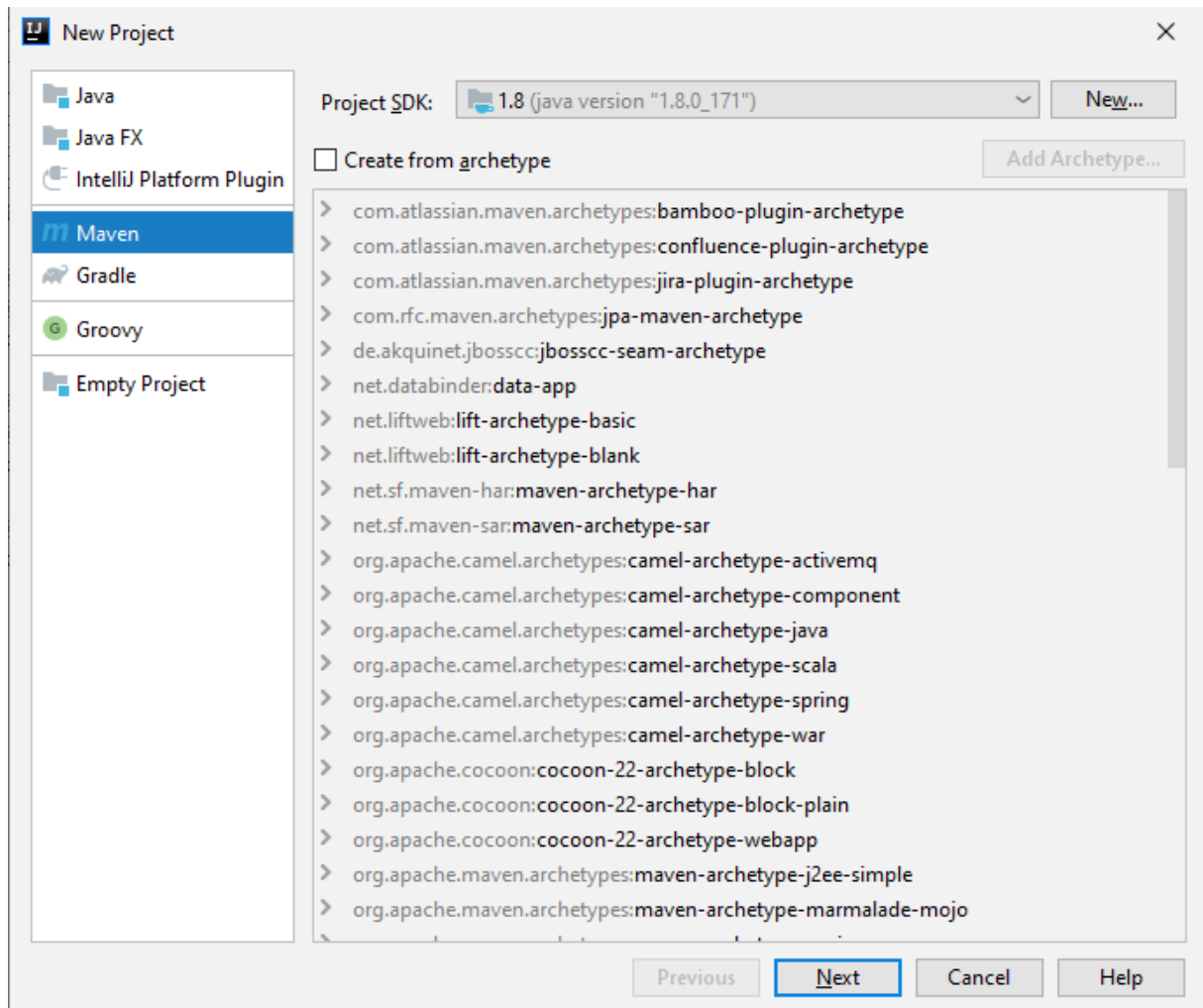


2. Należy utworzyć nowy projekt *maven* w środowisku programistycznym IntelliJ.  
(*maven* do pobrania jest na stronie: <https://maven.apache.org/download.cgi>,  
instrukcja instalacji znajduje się na stronie: <https://maven.apache.org/install.html>).

W tym celu wybieramy kolejno:

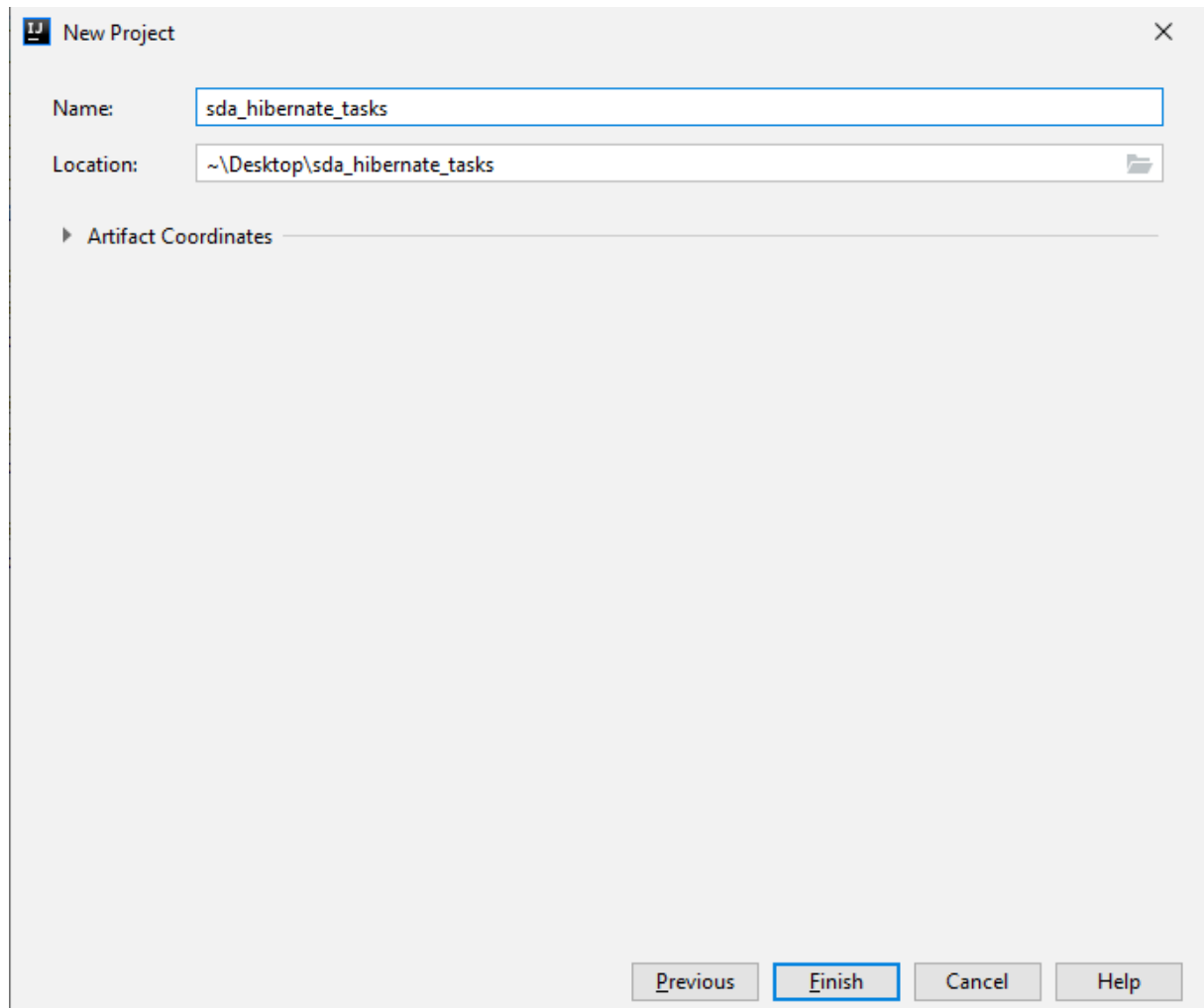
*File -> New -> Project*

Wybieramy projekt *maven*:



Klikamy dalej *Next* oraz na kolejnym ekranie wpisujemy nazwę aplikacji (dowolna, tj. według własnych upodobań) oraz wskazujemy lokalizację, w której *IntelliJ* utworzy nowy projekt oraz będą przechowywane źródła aplikacji.

Przykładowo:



Klikamy *Finish*.

W utworzonym pliku *pom.xml* dodajemy następujące zależności (sekcja *dependencies*, pod sekcją *version*):

```
<dependencies>
  <!-- Sterownik JDBC -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.21</version>
  </dependency>

  <!-- Framework hibernate -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.19.Final</version>
  </dependency>

  <!-- Biblioteka do logowania -->
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
  </dependency>

  <!-- Biblioteka do automatycznego generowania kodu -->
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Możliwości biblioteki *logback* oraz *lombok* zostaną przedstawione w trakcie trwania zajęć. W celu prawidłowego zaimportowania odpowiednich zależności należy wybrać prawym przyciskiem kliknąć na plik *pom.xml*, następnie z menu kontekstowego wybrać *Maven*, a następnie *Reimport*.

W przypadku korzystania z biblioteki *lombok* zalecana jest instalacja odpowiedniego pluginu w środowisku *Intellij*. W tym celu wybieramy: *File -> Settings (Ctrl + Alt + S) ...*, W oknie dialogowym, po lewej stronie wybieramy zakładkę *Plugins*. W wyszukiwarce wpisujemy: *Lombok*. Instalujemy plugin jeżeli nie został zainstalowany.

W katalogu *resources* nowo utworzonego projektu tworzymy plik *hibernate.cfg.xml* definiujący m.in. sposób połączenia z bazą danych.

Przykładowa postać pliku:

```
<?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>
        <!-- ustawienia połączenie do bazy danych -->
        <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost:3306/sda_hibernate?serverTimezone=UTC
</property>
        <property name="connection.username">root</property>
        <property name="connection.password">admin</property>

        <!-- SQL dialekt -->
        <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>

        <!-- Logowanie zapytań SQL -->
        <property name="hibernate.show_sql">true</property>
        <!-- Formatowanie zapytań SQL -->
        <property name="hibernate.format_sql">true</property>

    </session-factory>
</hibernate-configuration>
```

Kolejno, w naszym projekcie tworzymy pakiet: *pl.sda.hibernate*.

W pakiecie tym, tworzymy klasę o nazwie *SDASessionFactory*, która będzie odpowiedzialna za tworzenie instancji klasy implementującej interfejs *SessionFactory*.

Przykładowa implementacja klasy:

```
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class SDASessionFactory {

    private static SessionFactory sessionFactory;

    SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure().build();
            Metadata metadata = new
MetadataSources(standardRegistry).getMetadataBuilder().build();

            sessionFactory = metadata.getSessionFactoryBuilder().build();
        }

        return sessionFactory;
    }
}
```

Następnie, tworzymy klasę o nazwie *SDATaskManager*, która będzie odpowiedzialna za tworzenie obiektów sesji (na podstawie fabryki sesji) oraz wywołanie odpowiednich zadań.

Przykład implementacji:

```
import lombok.extern.slf4j.Slf4j;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

@Slf4j
public class SDATaskManager {

    public static void main(String... args) {
        try (SessionFactory sessionFactory = new
SDASessionFactory().getSessionFactory();
            Session session = sessionFactory.openSession()) {

            Log.info("Uzyskano obiekt sesji!");
        }
    }
}
```

Uruchamiamy aplikację (Ctrl + Shift + F10) lub klikając prawym przyciskiem myszy na klasę *SDATaskManager* i wybierając z menu kontekstowego *Run 'SDATaskManager.main()'*.

Przykładowe logi uzyskane po uruchomieniu aplikacji:

```
21:27:02.848 [main] DEBUG org.hibernate.internal.SessionFactoryRegistry - Not binding SessionFactory to JNDI, no JNDI name configured
```

```
21:27:02.898 [main] DEBUG org.hibernate.stat.internal.StatisticsInitiator - Statistics initialized [enabled=false]
```

```
21:27:02.900 [main] INFO pl.sda.hibernate.SDATaskManager - Uzyskano obiekt sesji!
```

21:27:02.901 [main] DEBUG org.hibernate.internal.SessionFactoryImpl - HHH000031: Closing

21:27:02.902 [main] DEBUG org.hibernate.type.spi.TypeConfiguration\$Scope - Un-scoping TypeConfiguration [org.hibernate.type.spi.TypeConfiguration\$Scope@70925b45] from SessionFactory [org.hibernate.internal.SessionFactoryImpl@716a7124]

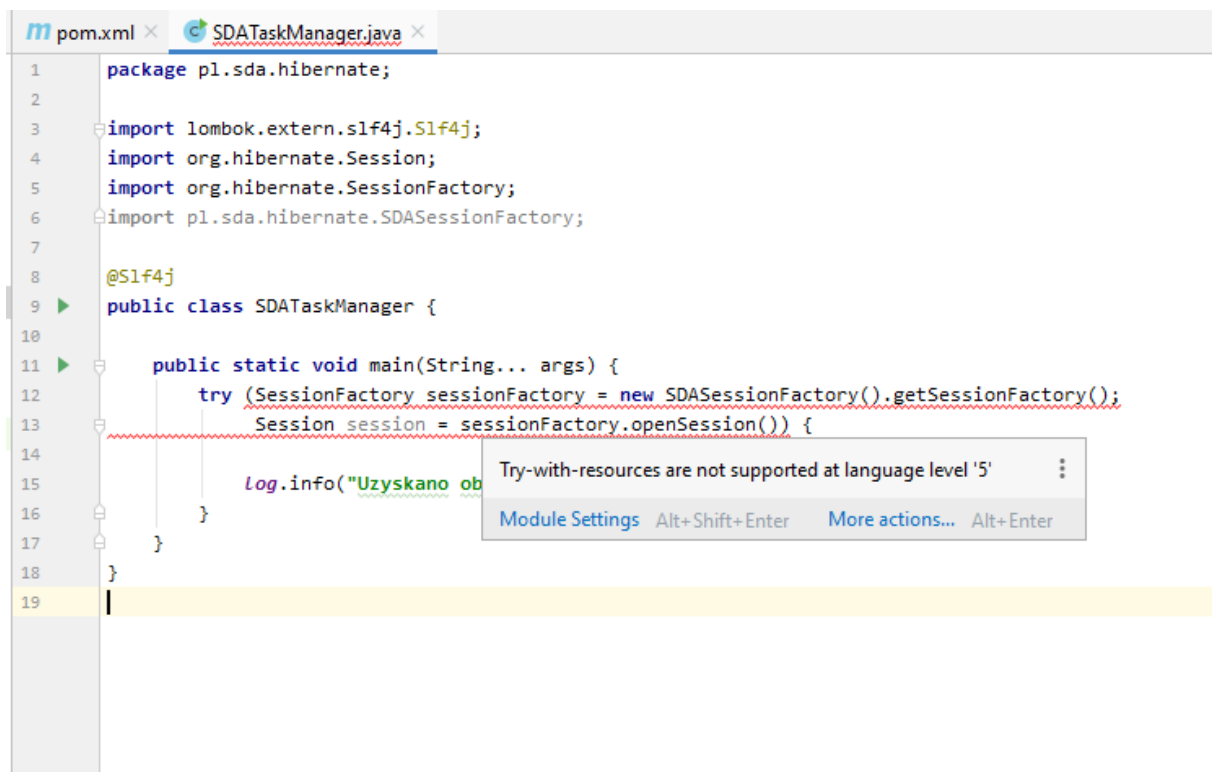
21:27:02.902 [main] DEBUG org.hibernate.service.internal.AbstractServiceRegistryImpl - Implicitly destroying ServiceRegistry on de-registration of all child ServiceRegistries

21:27:02.902 [main] INFO org.hibernate.orm.connections.pooling - HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/sda\_hibernate?serverTimezone=UTC]

21:27:02.903 [main] DEBUG org.hibernate.boot.registry.internal.BootstrapServiceRegistryImpl - Implicitly destroying Boot-strap registry on de-registration of all child ServiceRegistries

Uwaga.

W niektórych sytuacjach może pojawić się błąd w postaci:

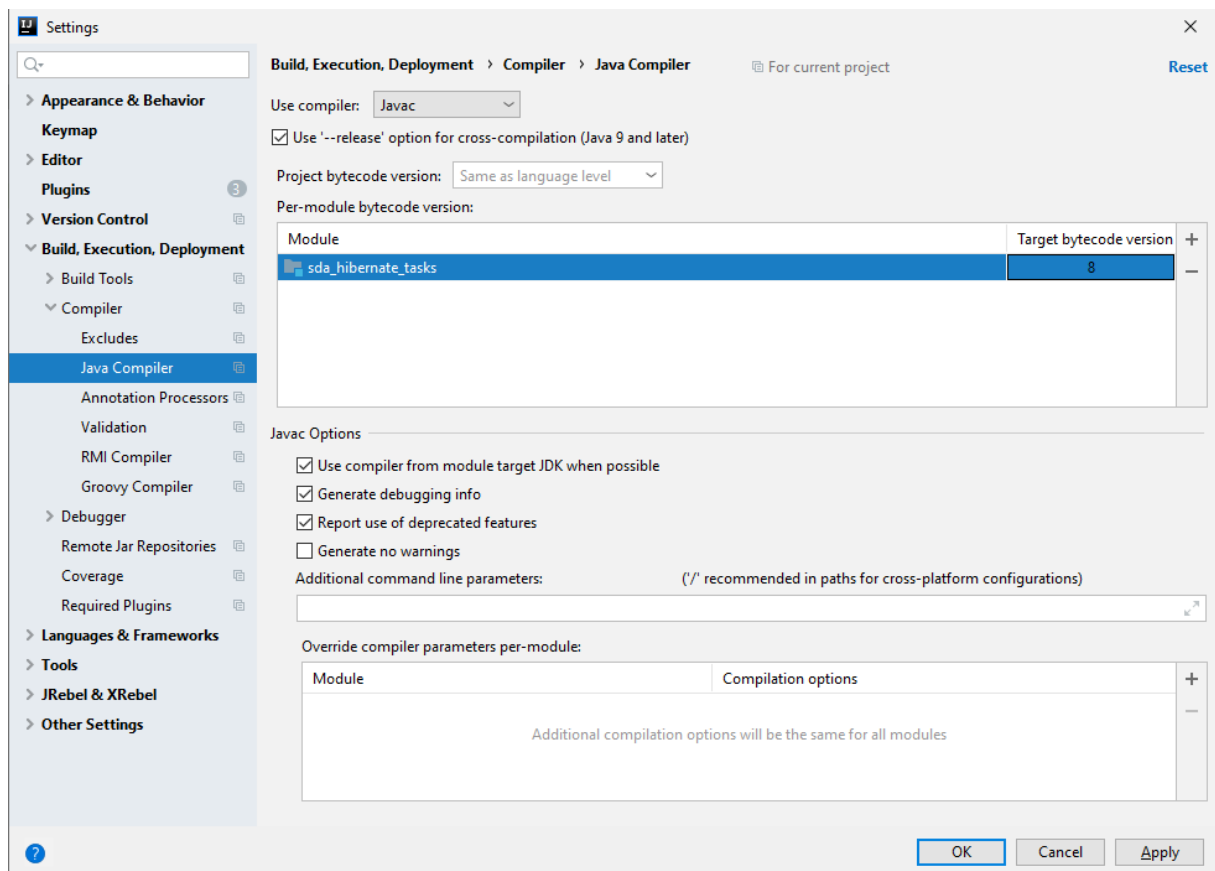


```
1 package pl.sda.hibernate;
2
3 import lombok.extern.slf4j.Slf4j;
4 import org.hibernate.Session;
5 import org.hibernate.SessionFactory;
6 import pl.sda.hibernate.SDASessionFactory;
7
8 @Slf4j
9 public class SDATaskManager {
10
11     public static void main(String... args) {
12         try (SessionFactory sessionFactory = new SDASessionFactory().getSessionFactory();
13             Session session = sessionFactory.openSession()) {
14
15             Log.info("Uzyskano ob");
16         }
17     }
18 }
19
```

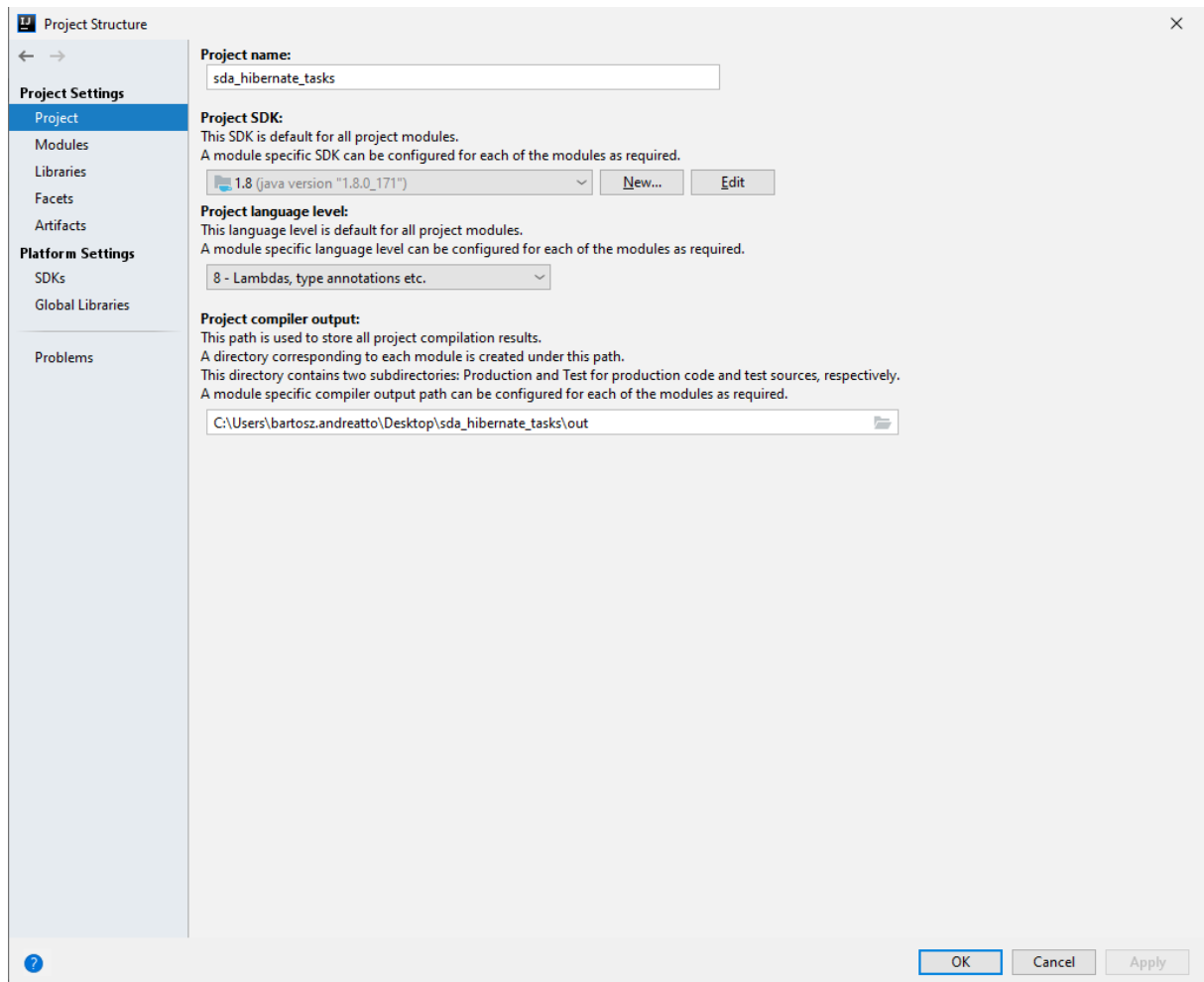
Try-with-resources are not supported at language level '5'

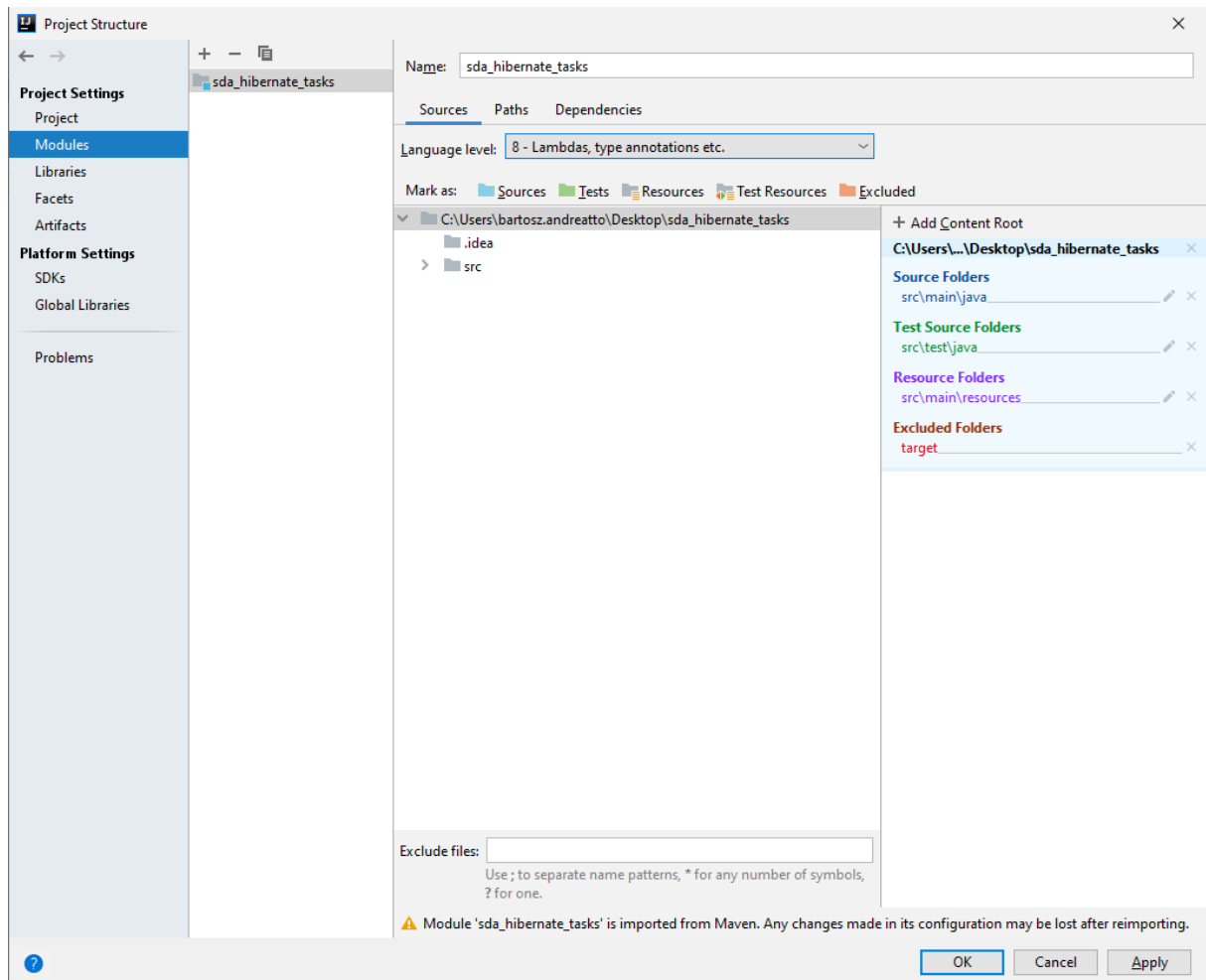
Module Settings Alt+Shift+Enter More actions... Alt+Enter

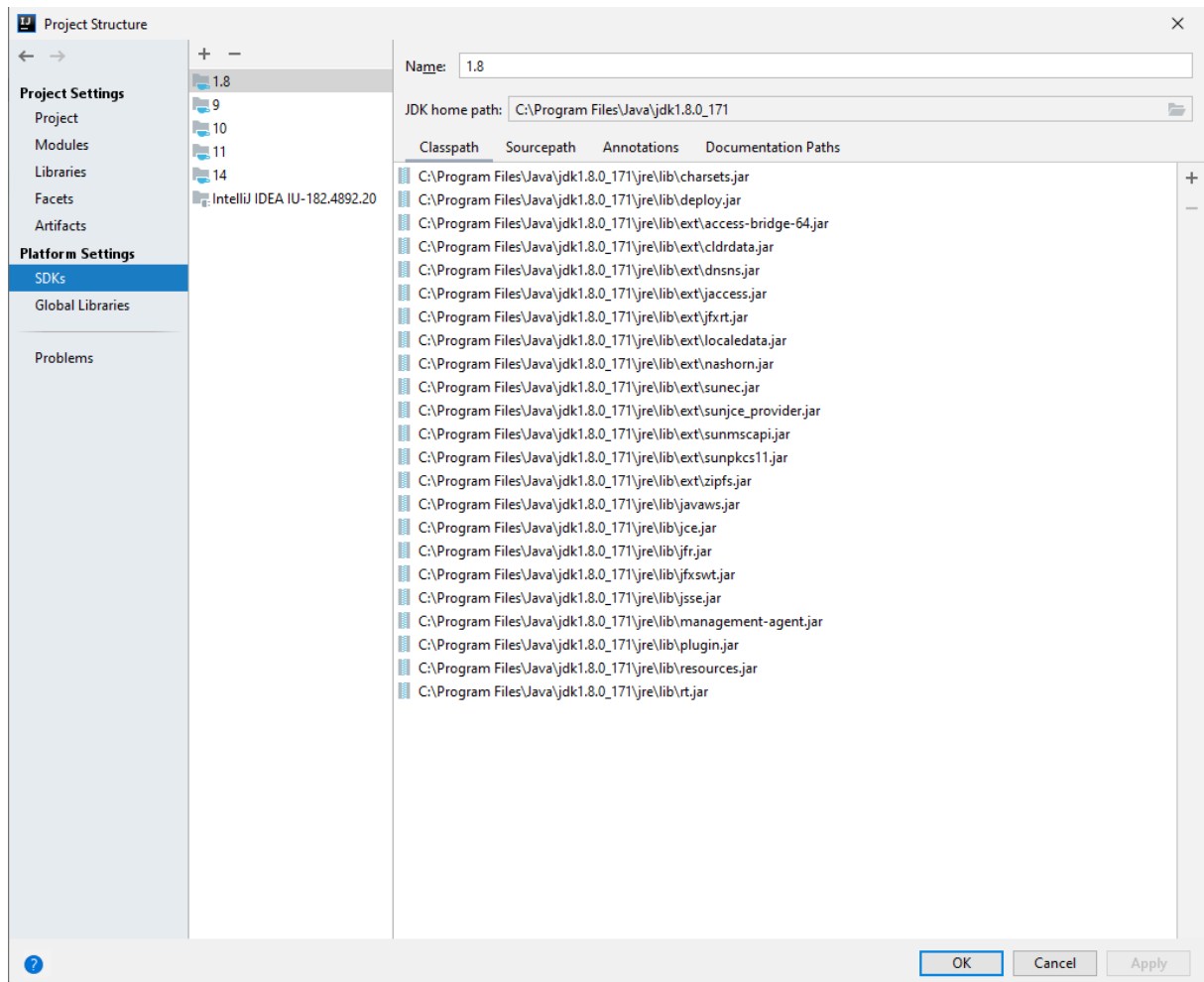
W zaistniałej sytuacji należy sprawdzić poniższe ustawienia:











3. Zadanie polega na pobraniu czytelnika o podanym identyfikatorze.

W tym celu należy utworzyć nowy pakiet *pl.sda.hibernate.model*

W pakiecie tym należy utworzyć klasę reprezentującą (odzworowującą) tabelę *Czytelnik*.

Dodatkowo, w pliku *hibernate.cfg.xml* należy dodać w sekcji *session-factory* następującą linię:

```
<mapping class="pl.sda.hibernate.model.Czytelnik"/>
```

Kolejno, tworzymy nowy pakiet *pl.sda.hibernate.tasks*. W pakiecie tym będą przechowywane odpowiednie klasy realizujące właściwe zadania.

W pakiecie tym tworzymy nową klasę nazwę *Task1*.

W klasie tej, implementujemy metodę o sygnaturze:

```
public void solve(Session session, Long id)
```

W ramach implementacji należy wykorzystać metodę *load* dostępną w klasie implementującej interfejs *Session*. Dodatkowo należy zadbać o warunki brzegowe, tj. sytuacje, w której nie zostanie odnaleziona encja dla podanego identyfikatora.

4. Wykorzystując odpowiednio przygotowane zapytanie HQL należy znaleźć oraz wypisać wszystkie kategorie książek.

5. Wykorzystując odpowiednio przygotowane zapytanie HQL należy odczytać identyfikator, tytuł oraz autora książek, których liczba stron jest większa lub równa wartości parametru przekazanego do

metody.

6. Zadanie polega na odczytaniu wybranych atrybutów (np. id, autor, kategoria, nazwa kategorii) książki, która należy do kategorii przekazanej jako argument zapytania (w postaci nazwy kategorii).
7. Zadanie polega na dodaniu nowego bibliotekarza. Realizując to zadanie należy koniecznie zwrócić uwagę na obsługę transakcji. Kolejno, należy porównać metody *save*, *persist*, *saveOrUpdate*.
8. Zadanie polega na aktualizacji numeru telefonu wybranego czytelnika. Należy zwrócić uwagę na metodę *save*, *update* i obsługę transakcji.
9. Zadanie polega na aktualizacji loginu oraz hasła bibliotekarza. Podejście opiera się na rozwiązaniu bazującym na obiektach niezarządzanych.
10. Zadanie polega na usunięciu bibliotekarza o wskazanym identyfikatorze. Rozważyć zastosowanie metody *remove* oraz *delete*. Zwrócić uwagę na obsługę transakcji. Rozważyć usuwanie obiektów zarządzanych i niezarządzanych.
11. W oparciu o zdefiniowane *NamedQuery* wyszukać wszystkie odebrane zamówienia (*data odbioru is not null*) danego czytelnika (uwzględniając login czytelnika przekazany jako parametr wywołania danej metody).
12. Zapisać w sposób kaskadowy nową kategorię (tzn. zapisując jednocześnie dwie nowe książki należące do danej kategorii).
13. Zadanie polega na wypisaniu danych administratora, realizując wyszukiwanie w oparciu o przekazany jako parametr wywołania login. Realizując zadania proszę uwzględnić metodę *setMaxResults()* dostępną w ramach interfejsu Query. Uwaga. Dane uwierzytelniające powinny być zamodelowane w postaci obiektu typu *Embedded*.
14. Zadanie polega na wyszukaniu książki (po numerze ISBN) oraz wypisaniu zamówień, w których brała udział. Wypisane zamówienia powinny być posortowane malejąco po dacie zamówienia. Zwrócić uwagę na prawidłowe modelowanie relacji. Wykorzystać adnotację *@OrderBy*.
15. Zadanie polega na pobraniu encji *@pl.sda.hibernate.model.Logowanie* i wypisaniu jej zawartości. Uwaga. Wyszukiwanie należy zrealizować w oparciu o klucz złożony (*IdClass* oraz *EmbeddedId*).
16. Zamodelować relację One2One łączącą czytelnika i adres. Wyszukać danego czytelnika wraz jego adresem. Dane zalogować.
17. Utworzyć repozytorium książek udostępniające takie funkcjonalności jak:
  - dodawanie książki
  - usuwanie książki
  - wyszukiwanie wszystkich książek
  - wyszukiwanie książki po nazwie (wykorzystać operator *like*)
  - wyszukiwanie książki po numerze ISBN (wykorzystać operator *like*)