Практическое задание №4.

Подготовительная работа.

Для выполнения данного задания необходимо установить IntelliJ Idea.

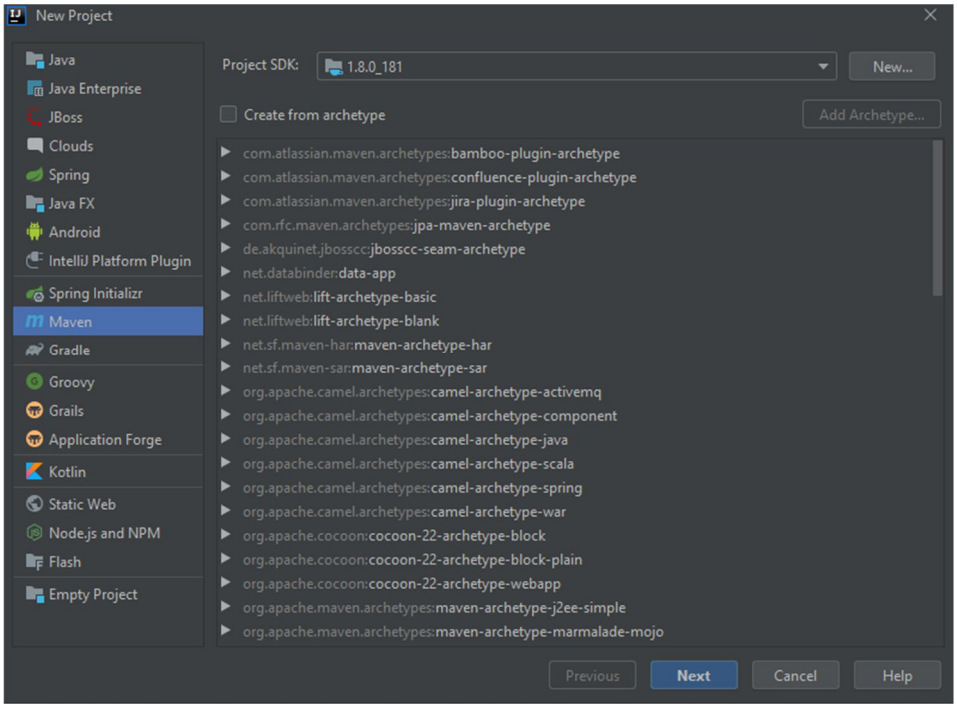
Получить данную IDE бесплатно для учебных целей можно воспользовавшись данной программой: https://www.jetbrains.com/shop/eform/students используя вашу электронную почту СФУ.

Кроме того, потребуется библиотека, содержащая CacheDriver, её вы можете найти в заголовке курса на e.sfu-kras.ru (jdbc-1.0).

Задание.

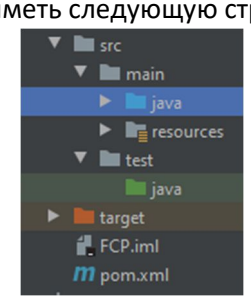
В рамках данной практической работы необходимо настроить связь между java приложением и базой данных, использую ORM прослойку Hibernate. Также необходимо протестировать данную связь путём исполнения стандартных запросов.

1. Создайте в IDE новый Maven проект:



   Рекомендуется использовать jdk версии 1.8 или выше.
   Укажите GroupId и ArtifactId.
   GroupId чаще всего содержит наименование организации или подразделения, и должно соответствовать правилам именавания java пакетов. Можете использовать для учебных заданий GroupId: edu.sfu.
   ArtifactId - это непосредственное название текущего проекта.

2. Сгенерированный проект должен иметь следующую структуру:



3. Файл Pom помимо служебной информации содержит так же информацию о зависимостях. Добавьте следующую зависимость:

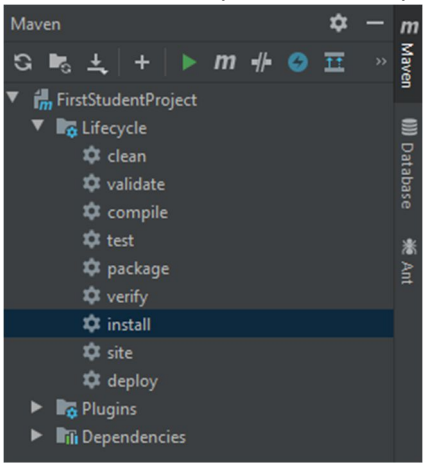
```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.11.Final</version>
</dependency>
```

   IDE предложит импортировать ее, подтвердите данное действие и дождитесь окончания загрузки. Если Maven используется на вашем компьютере впервые – будет сгенерирован локальный репозиторий .m2, который будет содержать скачанные библиотеки, которые в дальнейшем можно будет использовать напрямую в других проектах.

4. Глобального репозитория для jdbc-cache нет, поэтому будем использовать локальный файл. Для этого поместите скачанный файл jdbc в каталог resources вашего проекта, и пропишите следующую зависимость:

```
<dependency>
    <groupId>com.intersystems</groupId>
    <artifactId>jdbc</artifactId>
    <version>1.0</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/src/main/resources/jdbc-1.0.jar</systemPath>
</dependency>
```

5. Повторите импорт или же запустите команду сборки проекта, для этого можно воспользоваться правой панелью Maven и запустить команду install.



6. После того, как все зависимости подгружены можно приступать к настройкам подключения. Для этого в resources создайте новый xml файл: **hibernate.cfg.xml**. Данный файл будет содержать все настройки подключения, а так же mapping классов для следующих работ.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.intersys.jdbc.CacheDriver</property>
        <property name="hibernate.dialect">org.hibernate.dialect.Cache71Dialect</property>
        <property name="hibernate.connection.url">jdbc:Cache://localhost:51773/NAMESPACE</property>
        <property name="hibernate.connection.username">user</property>
        <property name="connection.password">password</property>
    </session-factory>
</hibernate-configuration>
```

Здесь user и password – созданный вами в первой работе пользователь IRIS. А NAMESPACE – ваша рабочая область IRIS.

7. Создайте новый java класс manager.DAO, для описания универсального взаимодействия с БД.
8. Создайте в данном классе следующие статические поля:

```
ThreadLocal<Session> session = new ThreadLocal<Session>();

SessionFactory sessionFactory = new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
```

Данные объекты потребуются нам для контроля сессий подключения.

9. Создайте в DAO статический метод getSession(), для получения новой сессии из sessionFactory. Данный метод должен проверять, существует ли открытая сессия в DAO, и в случае, если ее нет – запрашивать новую сессию, помещать ее в статическое поле session и возвращать, как результат выполнения метода.
10. Создайте в DAO статические методы **begin** и **commit**. Первый должен получать сессию и начинать транзакцию (*beginTransaction()*), второй должен подтверждать ранее открытую транзакцию у текущей сессии (*getTransaction().commit()*).
11. Создайте новый класс TestSrvs в пакете services. Данный класс должен являться наследником DAO. Реализуем в нем тестовый метод getName:

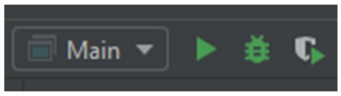
```
String query = "SELECT field FROM pkg.table WHERE ID=:id";
Query q = getSession().createSQLQuery(query)
        .setInteger( name: "id",id);
return (String) q.list().get(0);
```

Где field – текстовое поле вашей таблицы, а pkg.table – ваша таблица из БД, а id – параметр функции.

12. Создайте класс Main. И реализуйте в нем точку входа.

```
public static void main(String[] args){
```

13. Вызовите в данном методе созданный ранее метод getName. Выведите результат в консоль через функцию System.oup.println (*быстрый набор – sout Ctr+пробел – поверьте, данная функция потребуется вам очень часто*).
14. Запустите программу использую зеленую кнопку слева от функции main – это автоматически создаст конфигурацию запуска, которую можно будет использовать дальше через стандартный запуск:



Результатом работы функции должна быть строка содержащее поле из записи в БД с заданным ID.

15. По аналогии с предыдущим методом самостоятельно реализуйте еще 2.
    1) Метод, который возвращает список строковых значений одного поля из заданного промежутка ID
    2) Метод, для создания новых записей в БД, для одной из таблиц. В отличии от предыдущих методов – данные метод должен быть обернут в пару begin() commit(), для того чтобы результат сохранился, а так же вместо метода list должен использовать метод executeUpdate.