Java Persistence Api (JPA) with Spring Data

Introduction

JPA is the standard Object Relational Mapping (ORM https://en.wikipedia.org/wiki/Object-relational_mapping) API. JPA is an abstraction of almost all the SQL server (Oracle, MySQL…).

Spring Data provide a familiar and consistent, Spring-based programming model for data access for relational and non-relational databases, map-reduce frameworks, and cloud-based data services: https://spring.io/projects/spring-data

Required software

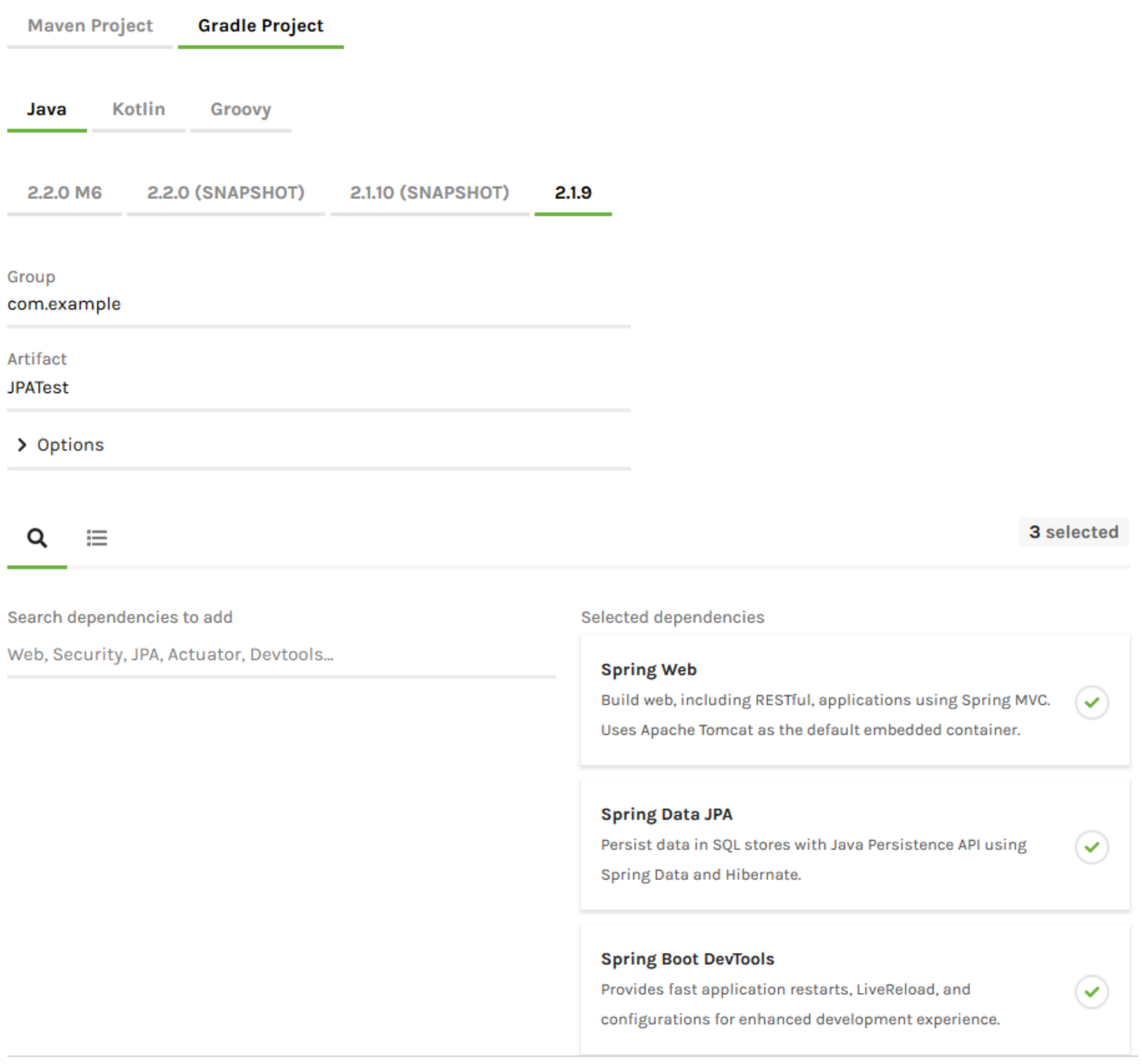
Java (JDK version)

Intellij or Eclipse

Gradle project creation

Use the Spring Initializer (https://start.spring.io/) to create a project:

- Don’t forget to choose Gradle Project
- Give it a name (Articfact)
- Add JPA as dependencies (library), Web (only for the database console) and DevTools (additional tools for cool edition)



Group
com.example

Artifact
JPATest

> Options

3 selected

Search dependencies to add
Web, Security, JPA, Actuator, Devtools...

Selected dependencies

Spring Web
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Download and unzip the project (outside the Eclipse workspace).

Then open the project (Intellij) or import it in Eclipse (File->Import->Gradle-> ... select the project directory).

The easiest way to deal with a database in to use an in memory one. Such a database can be H2. In order to use it edit the file build.gradle an add H2 in the dependencies:

```
dependencies {
	compile('org.springframework.boot:spring-boot-starter-web')
	implementation('org.springframework.boot:spring-boot-starter-data-jpa')
	runtime('org.springframework.boot:spring-boot-devtools')
	runtime("com.h2database:h2")
	testImplementation('org.springframework.boot:spring-boot-starter-test')
}
```

Don't forget to update your project (Intellij discovers any change in your project so just click on the Gradle elephant). In Eclipse, inside the File menu select gradle, then update the project.

Import the project under Eclipse: File->Import->General-> Existing project into workspace ... select the project directory

To enable the H2 console just add

```
spring.h2.console.enabled=true
```

into the Spring configuration file application.properties (into src/main/resources).

Then launch the main program: /src/main/java/package…/*Application.java

Verify the database content using the H2 web console: http://localhost:8080/h2-console

Then modify the JDBC URL with the good one provided into the Eclipse or Intellij console. Here is an example:

```
H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:dca7f3e9-95e2-425d-9ccf-c0aa71824aa1'
```

Create a persistent class

```java
package com.example.MaDemo;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Car {

    String plateNumber;
    long key;

    public String getPlateNumber() {
        return plateNumber;
    }

    public void setPlateNumber(String plateNumber) {
        this.plateNumber = plateNumber;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public long getKey() {
        return key;
    }

    public void setKey(long key) {
        this.key = key;
    }
}
```

Create a repository

```java
package com.example.MaDemo;

import org.springframework.data.repository.CrudRepository;

public interface CarRepository extends CrudRepository<Car, Long> {
}
```

Update the constructor of the web service class

```
@RestController
public class MonService {

    CarRepository carRepository;

    @Autowired
    public MonService(CarRepository carRepository){
        System.out.println(carRepository);
        this.carRepository = carRepository;
    }
```

Create a metho for saving a car

```
@PostMapping("/cars")
public void addCar(@RequestBody Car car){
    carRepository.save(car);
}
```

A JPA project example

An example of a JPA project using Spring Data is given: https://github.com/charroux/JPAExample/

It contains two persistent classes:
https://github.com/charroux/JPAExample/blob/master/src/main/java/com/efrei/JPAExample/City.java

https://github.com/charroux/JPAExample/blob/master/src/main/java/com/efrei/JPAExample/Person.java

An access to the database:
https://github.com/charroux/JPAExample/blob/master/src/main/java/com/efrei/JPAExample/CityRepository.java

A main program:

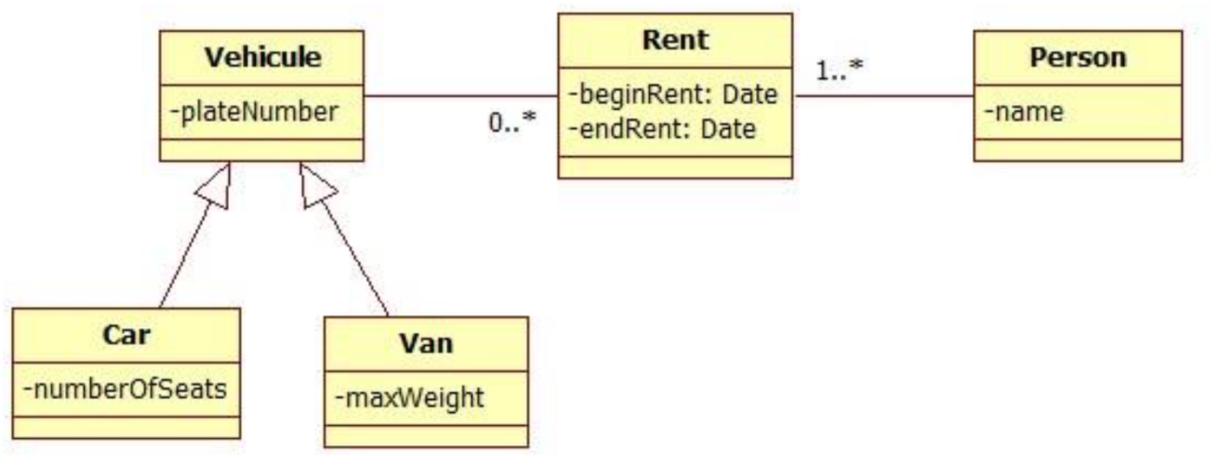
https://github.com/charroux/JPAExample/blob/master/src/main/java/com/efrei/JPAExample/JpaExampleApplication.java

There is also a Rest web service:

https://github.com/charroux/JPAExample/blob/master/src/main/java/com/efrei/JPAExample/RestWebService.java

Work to be done

Write the classes represented into the above class diagram:



To persist data into a database you must choose which format will be used in the dates. See http://docs.oracle.com/javaee/6/api/javax/persistence/Temporal.html.

An easy way to convert a String to a java.util.Date:

```
String pattern = "yyyy-MM-dd";
SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);

Date date = simpleDateFormat.parse("2018-09-09");
```

Write a main program that persist and retrieve objects into the database.

How can you use a single instruction storing a graph of objects?