DSD Lab 11: Spring databases (max: 12p)

In TASK 1, continue your ContactBook solution from last week. In TASK 2, use the Simple Spring Maven project template. Use **H2 database** in both tasks.

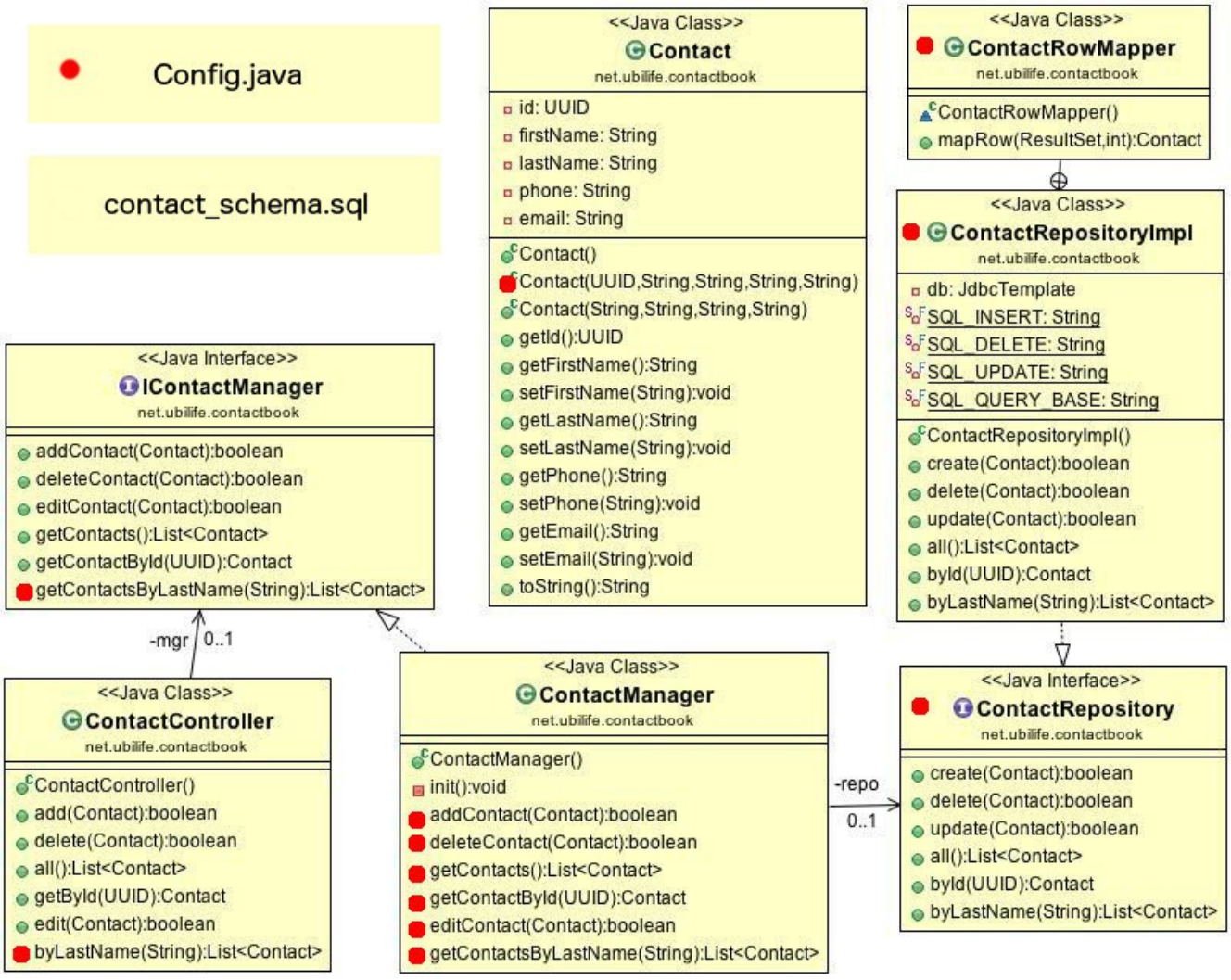
- IMPORTANT: modify pom.xml
  - Under <properties>, change **<org-springframework-version>** to the latest version **4.3.4.RELEASE**
  - Add these dependencies:
    - **com.h2database:h2:1.4.189**
    - **org.springframework:spring-jdbc:4.3.4.RELEASE** (TASK 1)
    - **org.springframework:spring-data-jpa:4.3.4.RELEASE** (TASK 2)

As usual, include your student ID in the project names and submit the projects ZIPped.

TASK 1: Contactbook with JdbcTemplate (6p)

Update the ContactBook application from last week lab to use **JdbcTemplate** and **H2 database** for storing contacts. If you want you can use the model solution from e-class (ContactBook_base.zip).

In the following diagram, red dots show the parts that should be updated/created:



You need a **Configuration file** for defining a data source and JdbcTemplate beans. The configuration file will scanned and loaded automatically as long as you use the @Configuration annotation.

Add the **schema script** (see **contact_schema.sql**) to your project's src/main/resources folder.

- IMPORTANT: Because H2 doesn't know how to generate UUID automatically, you should generate the id (UUID) in Contact's default constructor.

ContactManager forwards all calls to autowired **ContactRepository**, which uses an autowired JdbcTemplate to perform database operations. The repository methods are:

- **create()**: add new contact to database. IMPORTANT: You don't need to use the KeyHolder technique when inserting a new Contact because the method returns boolean.
- **delete()**: delete existing contact.
- **update()**: update existing contact.
- **all()**: get all contacts from database
- **byId()**: get one contact by id
- **byLastName()**: get contacts by lastname

SQL string constants in ContactRepository are defined as:

```
SQL_INSERT
       "insert into contact (id, firstName, lastName, phone, email) values (?, ?, ?, ?, ?)";

SQL_DELETE
       "delete from contact where id = ?";

SQL_UPDATE
       "update contact set firstName = ?, lastName = ?, phone = ?, email = ? where id = ?";

SQL_QUERY_BASE
       "select * from contact";    // add "where" constraints to this base string
```

ContactController needs a new method for getting contacts by lastname. You can choose the endpoint and the way of parameter delivery (POST, GET request parameter, or GET path parameter).

Tip: Convert UUID string to UUID object when you receive it from the result set in row mapper:

```
       UUID id = UUID.fromString(rs.getString("id"));
```

TASK 2: Car database with JPA and query methods (6p)

Define a **Car entity** with the following fields:

- Long id (primary key, auto-increment)
- String maker
- String model
- int makeYear
- int maxSpeed
- LicenseInfo (*Embedded*)

Define also a **LicenceInfo *Embeddable*** that has the following fields:

- String ownerName
- String ownerEmail
- Date registrationDate
- String licencePlateNumber

Create a **CarRepository interface** that extends JpaRepository. Define these **query methods**:

- Get all cars and order them by Maker in ascending order
- Get all cars owned by a name given as parameter.
- Get all cars made by a maker given as parameter.
- Get all cars made between two years given as parameters.
- Get one car that has the licence plate number given as parameter.

Create a standalone Java application to test your repository methods (add, edit, delete, find one/all, your query methods)

Tip 1: Here's how to define a date field in an entity with @Temporal annotation:

```
@Temporal(TemporalType.DATE)
private Date birthday;
```

Tip 2: You can write a query method for accessing an embedded object's fields like this:

```
findBy[EmbeddedObject][FieldInEmbeddedObject](String value);
```

Example:

```
@Embeddable
public class Address {
	private String streetAddress;
	...
}
```

```
public class Person {
	@Embedded
	private Address address;
	...
}
```

```
// A query method for finding a person by streetaddress
Person findByAddressStreetAddress(String street);
```