Web Frameworks: Spring

An Introduction

DI example

- Within a Controller

- `@Autowired`
- `CarService service;`

```
public interface VehicleService {

  public String transport();

}
```

```
@Service
public class CarService implements VehicleService {

  @Override
  public String transport(){
      return "Car transport";
  }

}
```

Managing Beans

Spring IoC container does the following

- to create a bean of *CarService*
- Assign the bean to the controller and add it to the *service* property

- To make an instance of any class that should be managed by Spring, it is necessary to add the annotation *@Component* over this particular class.
- This way Spring can manage this dependence, which means that Spring detects this as a bean, or an object managed by the Spring IoC Container.
- Spring *@Service* annotation is a specialization of *@Component* annotation. It is used to mark the class as a service provider
- The *@autowired* annotation is injecting *CarService* object into the property named *service*.

```java
public interface VehicleService {

  public String transport();

}
```

- Within a Controller

- `@Autowired`
- `VehicleService service;`

- When loaded, Spring will start looking for im-plementations of this interface, and since *CarService* class implements *VehicleService* interface, Spring will find this implementation and it will inject instance of appropriate class.

```java
@Service
public class CarService implements VehicleService {

  @Override
  public String transport(){
      return "Car transport";
  }

}
```

```java
@Service
public class TruckService implements VehicleService{

  @Override
  public String transport() {
    return "Truck transport";
  }

}
```

Linking services

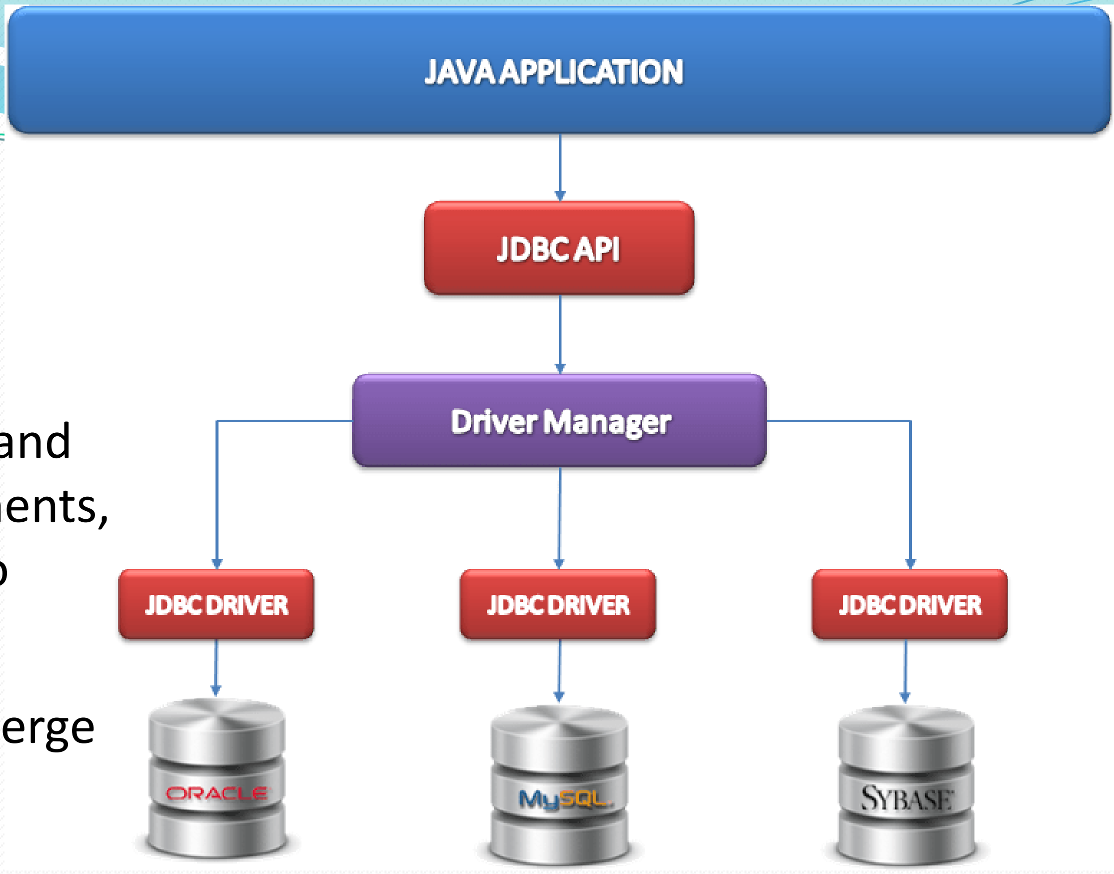
- First approach would be renaming a particular property, named *service* to the name of the concrete service, *carService*
  - `@Autowired`
  - `VehicleService carService;`
  - Spring detects that the name of the instance is *carService*, and that it is an implementation of *VehicleService*, so Spring is able to inject it
- Another approach in resolving this issue is adding *@Qualifier* annotation and passing the name of the bean that needs to be injected
  - `@Qualifier(value = "truckService")`
- DI can also be implemented through both constructor and set methods
- Performance cost of wiring beans is usually located in the start-up phase of application

Dependency Injection Pros & Cons

- Pros
  - Loosely Coupled
  - Increases Testability
  - Separates components cleanly
  - Allows for use of Inversion of Control Container
- Cons
  - Increases code complexity
  - Developers learning time increases
  - Can Complicate Debugging at First
  - Complicates following Code Flow

R. Laigner, D. Mendonça, A. Garcia, M. Kalinowski, Cataloging dependency injection anti-patterns in software systems, Journal of Systems and Software, Volume 184, 2022,111125,ISSN 0164-1212, https://doi.org/10.1016/j.jss.2021.111125.

Spring Data Project



- In JDBC, programmers had to download the correct drivers and connection strings, open and close connections, SQL statements, result sets, and transactions, and convert from result sets to objects
- ORM (object-relational mapping) frameworks started to emerge to manage these tasks
    - Hibernate
    - Castor XML
- They allowed you to identify the domain classes and create XML that was related to the database's tables.
- Spring utilizes such frameworks by following the *template design pattern*.

Spring Data Project

- It allowed you create an abstract class that defined ways to execute the methods
  - It also created the database abstractions that allowed you to focus only on your business logic.
- the Spring Framework relies on several interfaces and classes (like the javax.sql.DataSource interface) to get information about the database you are going to use, how to connect to it (by providing a connection string), and its
- Normally, the DataSource interface requires
  - the Driver class, the JDBC URL,
  - A username, and
  - a password to connect to the database.
- It left all the hard lifting to the Spring Framework, including handling connections (open, close, and pooling), transactions, and the way you interact with the frameworks.
- The Spring Data project is the umbrella for several additional libraries and data frameworks, which makes it easy to use data access technologies for relational and non-relation databases (a.k.a. NoSQL).

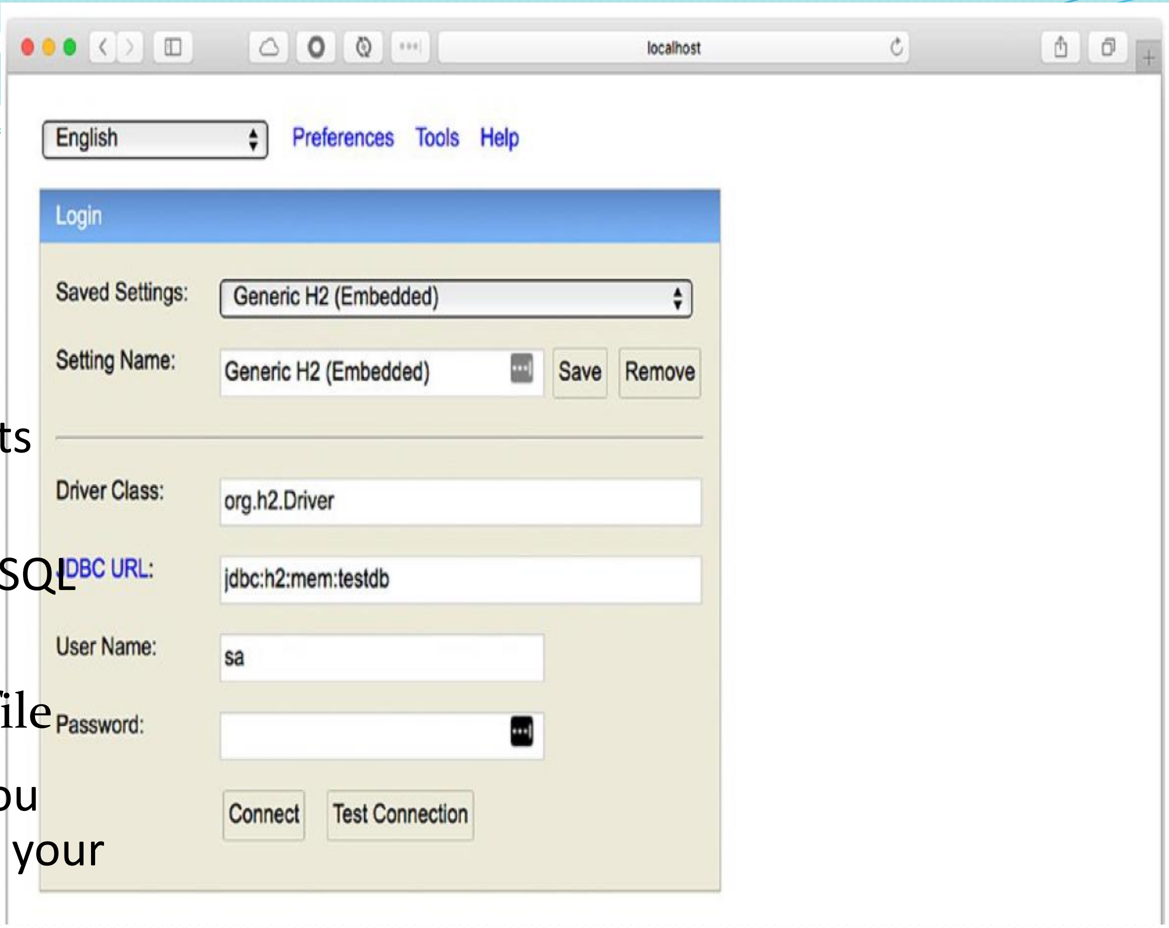
Spring Data Project

- Some of the Spring Data features.
  - • Support for cross-store persistence
  - • Repository-based and custom object-mapping abstractions
  - • Dynamic queries based on method names

-
  ```
  String SQL_INSERT = "insert into todo (id, description, created, modified, completed) values (:id,:description,:created,:modified,:completed)";
  ```
  - • Support for Spring MVC controllers
  - • Events for transparent auditing (created, last changes)

Connecting to H2

- Spring Boot uses auto-configuration to set sensible defaults when it finds that your application has a JDBC JAR
- Spring Boot auto-configures the datasource based on the SQL driver in your classpath
- No need to mention any data source in the properties file
- If you want to use JDBC in your Spring Boot application, you need to add the spring-boot-starter-jdbc dependency and your SQL driver
- ***To enable H2 console***
  - ***src/main/resources/application.properties***
  - spring.h2.console.enabled=true
- *http://localhost:8080/h2-console*



Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

User Name: sa

Password:

Connecting MySql

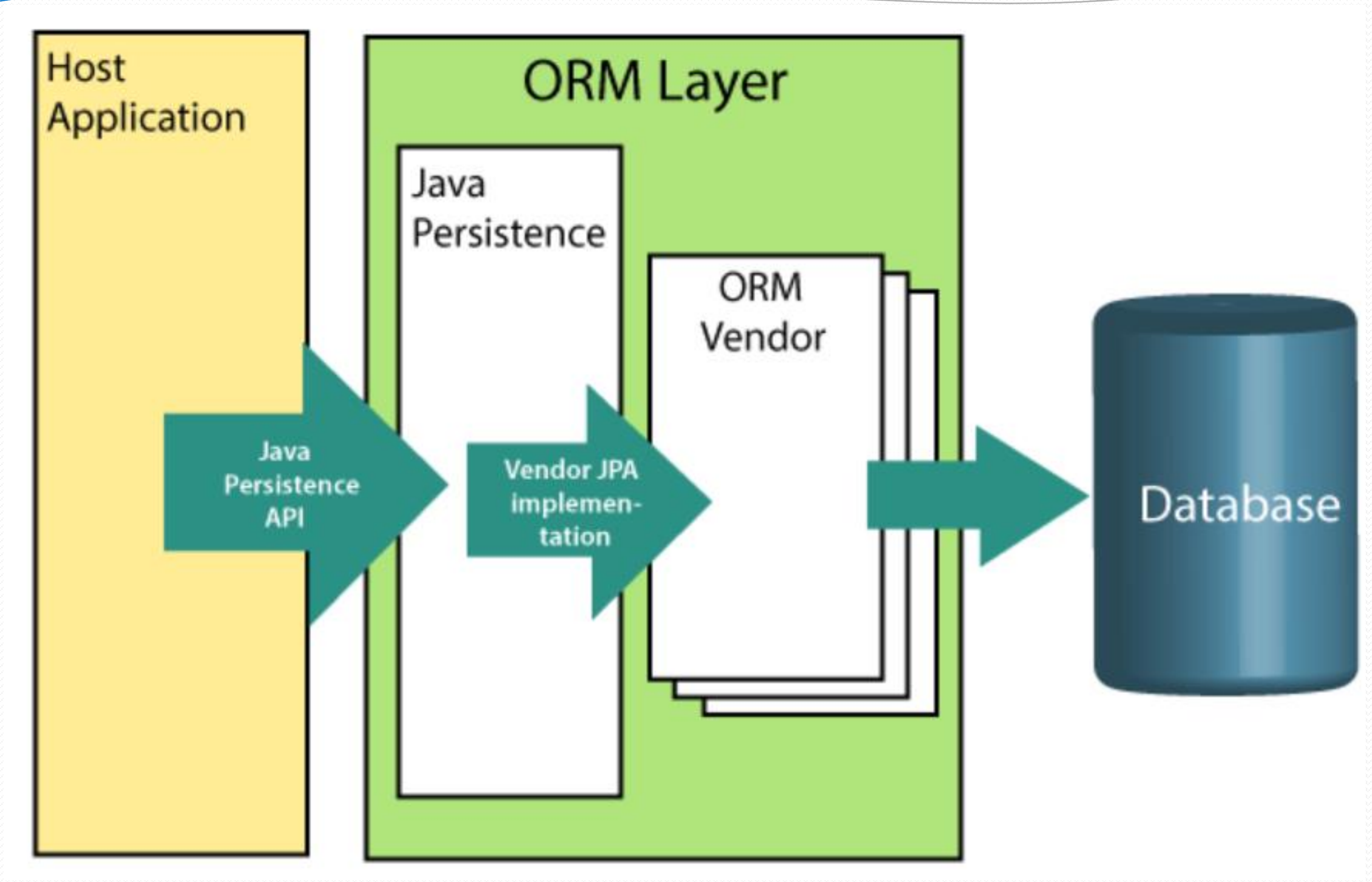
- spring.datasource.url=jdbc:mysql://localhost:3306/testconnect
- spring.datasource.username=root
- spring.datasource.password=1234
- spring.datasource.driver-class-name= com.mysql.cj.jdbc.Driver

Spring-JDBC

- Another feature that Spring Boot brings to data apps is that if you have a file named *schema.sql*, *data.sql*, *schema-<platform>.sql*, or *data-<platform>.sql* in the classpath, it initializes your database by executing those script files
- The file should be kept in src/main/resources/
- So, If you want to use JDBC in your Spring Boot application, you need to add the spring-boot-starter-jdbc dependency and your SQL driver.
- Spring provides a JDBCTemplate class
- The JdbcTemplate class offers you a lot of possibilities to interact with any database engine
  - You can use NamedParameterJdbcTemplate (a JdbcTemplate wrapper) to provide named parameters (:parameterName), instead of the traditional JDBC "?" placeholders.
  - It provides wrapper for ResultSet to access each row and many more

Spring Data JPA

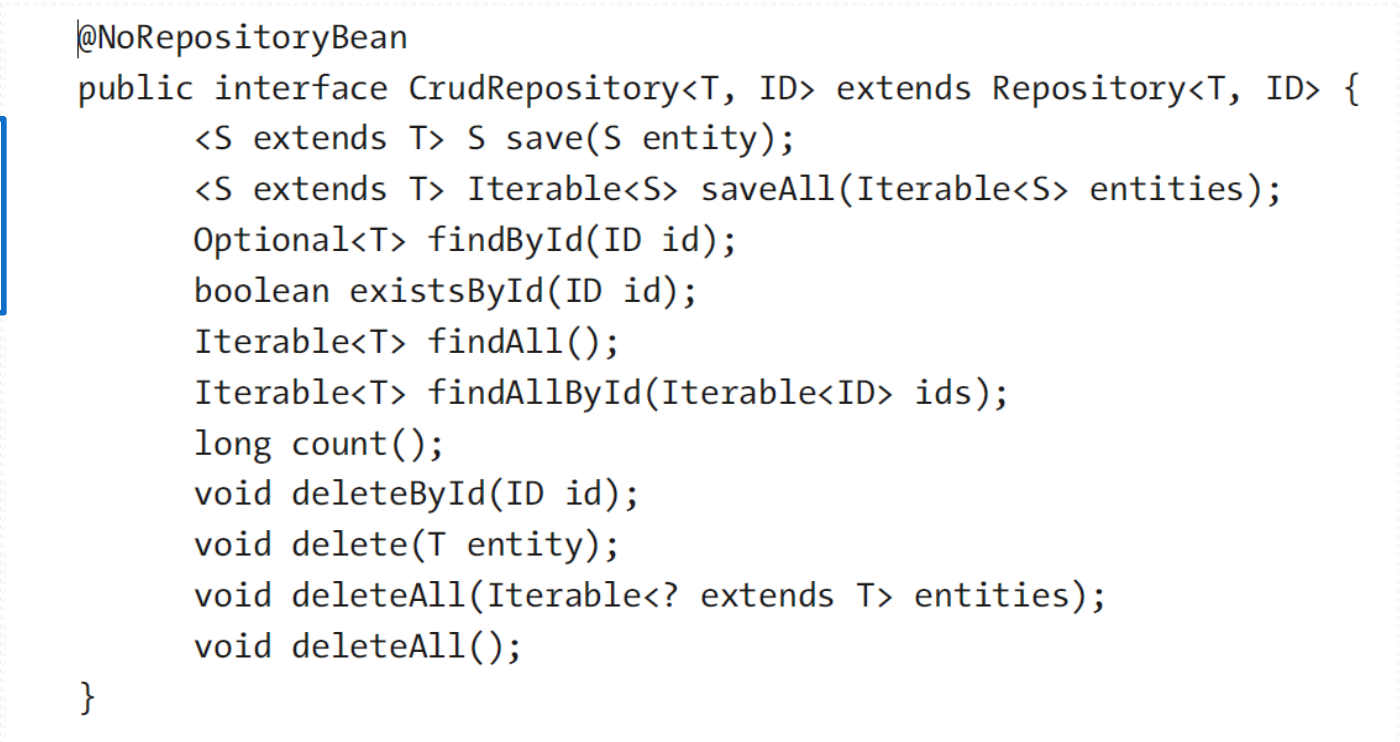
- The JPA (Java Persistence API) provides a POJO persistence model for object-relational mapping
- It follows the *aggregate root* concept
  - Support for repositories (a concept from *Domain-Driven Design*)
- Implementing data access can be a hassle because we need to deal with connections, sessions, exception handling, and more, even for simple CRUD operations
- That’s why the Spring Data JPA provides an additional level of functionality: creating repository implementations directly from interfaces and using conventions to generate queries from method names
- Pagination, sort, dynamic query execution support.
- Support for @Query annotations
- JavaConfig based repository configuration by using the @EnableJpaRepositories annotation.


Host Application
ORM Layer
Java Persistence
ORM Vendor
Java Persistence API
Vendor JPA implemen-tation
Database

Repository Concept

- The most compelling feature of Spring JPA is the ability to create repository implementations automatically, at runtime, from a repository interface.
- We only need to create an interface that extends from a Repository<T,ID>, CrudRepository<T,ID>, or JpaRepository<T,ID>
- The JpaRepository interface offers not only what the CrudRepository does, but also extends from the PagingAndSortingRepository interface that provides extra functionality

the T means the entity (your domain model class) and the ID, the primary key that needs to implement Serializable.



```
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
      <S extends T> S save(S entity);
      <S extends T> Iterable<S> saveAll(Iterable<S> entities);
      Optional<T> findById(ID id);
      boolean existsById(ID id);
      Iterable<T> findAll();
      Iterable<T> findAllById(Iterable<ID> ids);
      long count();
      void deleteById(ID id);
      void delete(T entity);
      void deleteAll(Iterable<? extends T> entities);
      void deleteAll();
}
```

Application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/testconnect
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

- Advanced translation of PersistenceExceptions to Spring DataAccessException
- Applying specific transaction semantics such as custom isolation level or transaction timeout

Application properties

Spring Boot provides properties that allow you to override defaults when using the Spring Data JPA. One of them is the ability to create the DDL (data definition language), which is turned off by default, but you can enable it to do reverse engineering from your domain model.

- Hibernate ddl auto (create, create-drop, update)
  ```
  spring.jpa.hibernate.ddl-auto=update
  ```

- none: The default for MySQL. No change is made to the database structure.
- update: Hibernate changes the database according to the given entity structures.
- create: Creates the database every time but does not drop it on close.
- create-drop: Creates the database and drops it when SessionFactory closes

-
  ```
  spring.jpa.show-sql=true
  ```
- The simplest way is to dump the queries to standard out

Java Persistence API

In a simple Spring app, you are required to use the @EnableJpaRepositories annotation that triggers the extra configuration that is applied in the life cycle of the repositories defined within of your application

The class that should persist in the database

```
@Entity
public class Person {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;

   private String firstName;
   private String lastName;
Getter and setter methods, constructors both versions }
```

Create the repository

- **public interface PersonRepository extends CrudRepository<Person, Long> {**
- `  List<Person> findByFirstName(String firstName);`
- `}`

- Invoke the database through dependency injection
- @RepositoryRestResource(collectionResourceRel = "people", path = "people")
  - The @RepositoryRestResource annotation is optional and is used to customize the REST endpoint
  - At runtime, Spring Data REST automatically creates an implementation of this interface.
  - collectionResourceRel -The rel value to use when generating links to the collection resource.
  - Path-The path segment under which this resource is to be exported.

- Then it uses the @RepositoryRestResource annotation to direct Spring MVC to create RESTful endpoints at /people