Integrating with Data Sources

1. Understanding Spring Data
2. Getting started with JPA
3. Defining JPA entity classes
4. Viewing database data

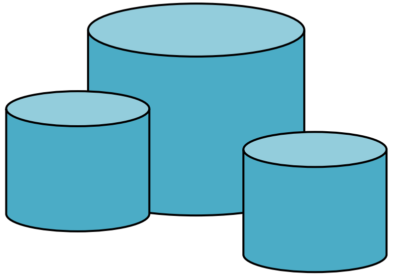
1. Understanding Spring Data

- Spring vertical data access APIs
- About Spring Data
- Adding the data source driver to the classpath

Pearson

Demo package: `demo.springdata`

Spring Vertical Data Access APIs



- Spring provides vertical APIs for data access
  - Many technologies, including JDBC, JPA, etc.
- Declarative transaction management
  - Transactional boundaries declared via configuration
  - Enforced by a Spring transaction manager
- Automatic connection management
  - Acquires/releases connections automatically

About Spring Data

- Spring Data supports many data access technologies
  - See https://spring.io/projects/spring-data

- Powerful repository and object-mapping abstractions

- Dynamic query creation from repository method names

Adding the Data Source Driver to the Classpath

- Add the appropriate Maven dependency for the type of data source you wish to access, e.g. H2:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

pom.xml

- H2 is an in-memory database
  - Created/dropped when app starts/ends
  - Very handy during development ☺️

Pearson

2. Getting Started with JPA

- Overview of JPA
- Important JPA concepts
- JPA dependency in Spring Boot
- Spring Boot autoconfiguration
- Customizing persistence properties

Overview of JPA

- JPA = Java Persistence API
    - A standard ORM (object/relational mapping) API
- JPA is a specification
    - Implemented by the Hibernate library
    - Also implemented by Java Enterprise Edition
- To use JPA in Spring:
    - Add the Hibernate library to your classpath, see later

Important JPA Concepts

- Entity class
  - A Java class, mapped to a relational database table
- Entity manager
  - Provides an API to fetch/save entities to a relational database
- Entity manager factory
  - Creates and configures an entity manager so it can connect to a relational database

JPA Dependency in Spring Boot

- To use JPA in a Spring Boot application, add the following dependency to your POM file:

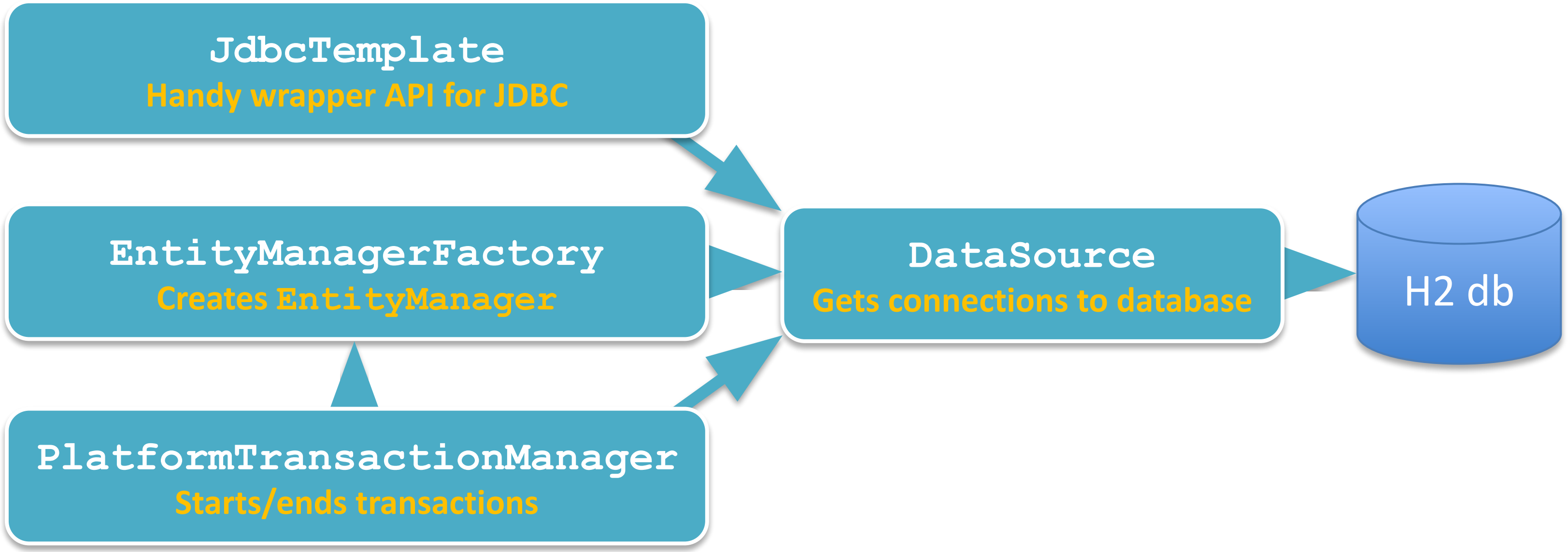
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

pom.xml

- This adds all the relevant Hibernate libraries to the classpath

Spring Boot Autoconfiguration

- Courtesy of the JPA dependency, Spring Boot creates several beans automatically in your application



Customizing Persistence Properties

- Spring Boot automatically sets persistence properties to connect to the in-memory H2 database:

```
spring.datasource.url=jdbc:h2:mem:<UUID>
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

application.properties

- You can customize persistence properties if you need to:

```
// Show SQL statements, nicely formatted.
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.format_sql=true

// Optionally, you can also ask JPA to output the value for SQL parameters.
logging.level.org.hibernate.SQL=debug
logging.level.org.hibernate.type.descriptor.sql=trace
```

application.properties

3. Defining JPA Entity Classes

- How to define an entity class
- Locating entity classes
- Seeding the database with data

How to Define an Entity Class

- You can define an entity class as follows:

```
import javax.persistence.*;

@Entity
@Table(name="EMPLOYEES")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long employeeId = -1;

    private String name;
    private String region;

    @Column(name="salary")
    private double dosh;

    // Plus constructors, getters/setters,
    // equals(), and hashCode()
}
```

Employee.java

Locating Entity Classes

- A Spring Boot app scans for entity classes when it starts
    - It looks in the main app class package, plus sub-packages
- You can tell it to look elsewhere, if necessary
    - Via `@EntityScan`

```
@SpringBootApplication
@EntityScan( {"myentitypackage1", "myentitypackage2"} )
public class Application {
    …
}
```

Seeding the Database with Data

- For convenience during development/testing, you can seed the database with some sample data

```
import org.springframework.jdbc.core.JdbcTemplate;
…

@Component
public class SeedDb {

    @Autowired
    JdbcTemplate jdbcTemplate;

    @PostConstruct
    public void init() {
        jdbcTemplate.update(
            "insert into EMPLOYEES(name,salary,region) values(?,?,?)",
            new Object[]{"James", 21000, "London"});
        …
    }
}                                                          SeedDb.java
```

4. Viewing Database Data

- Overview
- Obtaining the database connection string
- Viewing the database data in the H2 console UI

Overview

- Most databases have a console UI to let you view data
  - To enable the H2 console UI, add these application properties:

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

application.properties

- The H2 console UI is a web endpoint
  - So, add this dependency in your POM:

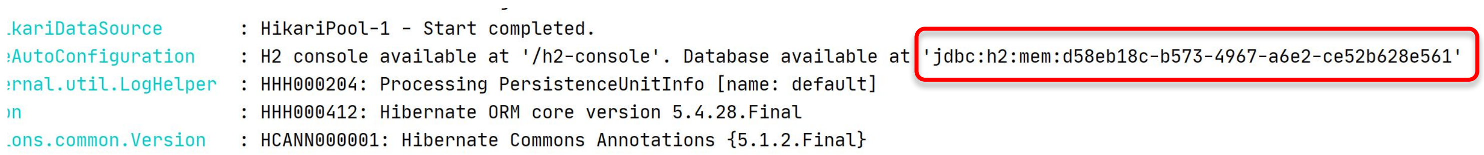
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

pom.xml

Pearson

Obtaining the Database Connection String

- When you run your app, you'll see a message that indicates the JDBC connection string for the database:



```
.kariDataSource       : HikariPool-1 - Start completed.
:AutoConfiguration    : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:d58eb18c-b573-4967-a6e2-ce52b628e561'
:rnal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
:n                    : HHH000412: Hibernate ORM core version 5.4.28.Final
.ons.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
```

- You can use this JDBC connection string to connect to the database in the H2 console UI …

Viewing the Database Data in the H2 Console UI

- To open the H2 console UI, browse to:
  - http://localhost:8080/h2-console

- To connect to the database, enter these details:
  - JDBC URL - as per previous slide
  - User name - `sa`
  - Password - leave blank

- You can then view tables in the database - cool!

Summary

- Understanding Spring Data
- Getting started with JPA
- Defining JPA entity classes
- Viewing database data

Exercise

- Define an entity class named `Car` with these fields:
  - `carId` (primary key)
  - `registrationNumber`
  - `make`
  - `model`
- Add some code in `SeedDb` to insert some cars
- Run the application and view the H2 console UI, to confirm the car data exists in the database