[Introduction](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Explain the need and benefit of ORM
  + ORM (Object-Relational Mapping), makes it easier to develop code that interacts with database, abstracts the database system, transactionality
    - ORM Pros and Cons - https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a
    - What is ORM? - https://en.wikipedia.org/wiki/Object-relational\_mapping

* Demonstrate the need and benefit of Spring Data JPA
  + Evolution of ORM solutions, Hibernate XML Configuration, Hibernate Annotation Configuration, Spring Data JPA, Hibernate benefits, open source, light weight, database independent query
    - With H2 in memory database - https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa/
    - With MySQL - https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa-mysql-example/
    - XML Configuration Example -https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm
    - Hibernate Configuration Example -https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm

* Explain about core objects of hibernate framework
  + Session Factory, Session, Transaction Factory, Transaction, Connection Provider
    - Hibernate Architecture Reference - https://www.tutorialspoint.com/hibernate/hibernate\_architecture.htm

* Explain ORM implementation with Hibernate XML Configuration and Annotation Configuration
  + XML Configuration - persistence class, mapping xml, configuration xml, loading hibernate configuration xml file; Annotation Configuration - persistence class, @Entity, @Table, @Id, @Column, hibernate configuration xml file Loading hibernate configuration and interacting with database get the session factory, open session, begin transaction, commit transaction, close session
    - XML Configuration Example - https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm
    - Hibernate Configuration Example - https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm

* Explain the difference between Java Persistence API, Hibernate and Spring Data JPA
  + JPA (Java Persistence API), JPA is a specification (JSR 338), JPA does not have implementation, Hibernate is one of the implementation for JPA, Hibernate is a ORM tool, Spring Data JPA is an abstraction above Hibernate to remove boiler plate code when persisting data using Hibernate.
    - Difference between Spring Data JPA and Hibernate - https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1
    - Intro to JPA - https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html

* Demonstrate implementation of DML using Spring Data JPA on a single database table
  + Hibernate log configuration and ddl-auto configuration, JpaRepsitory.findById(), defining Query Methods, JpaRespository.save(), JpaRepository.deleteById()
    - Spring Data JPA Ref Repository methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#repositories.core-concepts
    - Query methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#repositories.query-methods

**Spring Data JPA - Quick Example**  
  
**Software Pre-requisites**

* MySQL Server 8.0
* MySQL Workbench 8
* Eclipse IDE for Enterprise Java Developers 2019-03 R
* Maven 3.6.2

**Create a Eclipse Project using Spring Initializr**

* Go to <https://start.spring.io/>
* Change Group as “com.cognizant”
* Change Artifact Id as “orm-learn”
* In Options > Description enter "Demo project for Spring Data JPA and Hibernate"
* Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
* Click Generate and download the project as zip
* Extract the zip in root folder to Eclipse Workspace
* Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
* Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

> mysql -u root -p

mysql> create schema ormlearn;

* In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

# Log pattern

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# Database configuration

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=root

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

* Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
* Include logs for verifying if main() method is called.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

  LOGGER.info("Inside main");

}

* Execute the OrmLearnApplication and check in log if main method is called.

SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. OrmLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.

**Country table creation**

* Create a new table country with columns for code and name. For sample, let us insert one country with values 'IN' and 'India' in this table.

create table country(co\_code varchar(2) primary key, co\_name varchar(50));

* Insert couple of records into the table

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

**Persistence Class - com.cognizant.orm-learn.model.Country**

* Open Eclipse with orm-learn project
* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.
* Include @Entity and @Table at class level
* Include @Column annotations in each getter method specifying the column name.

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="country")

public class Country {

  @Id

    @Column(name="code")

    private String code;

    @Column(name="name")

    private String name;

// getters and setters

  // toString()

}

*Notes:*

* @Entity is an indicator to Spring Data JPA that it is an entity class for the application
* @Table helps in defining the mapping database table
* @Id helps is defining the primary key
* @Column helps in defining the mapping table column

**Repository Class - com.cognizant.orm-learn.CountryRepository**

* Create new package com.cognizant.orm-learn.repository
* Create new interface named CountryRepository that extends JpaRepository<Country, String>
* Define @Repository annotation at class level

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**Service Class - com.cognizant.orm-learn.service.CountryService**

* Create new package com.cognizant.orm-learn.service
* Create new class CountryService
* Include @Service annotation at class level
* Autowire CountryRepository in CountryService
* Include new method getAllCountries() method that returns a list of countries.
* Include @Transactional annotation for this method
* In getAllCountries() method invoke countryRepository.findAll() method and return the result

**Testing in OrmLearnApplication.java**

* Include a static reference to CountryService in OrmLearnApplication class

private static CountryService countryService;

* Define a test method to get all countries from service.

    private static void testGetAllCountries() {

        LOGGER.info("Start");

        List<Country> countries = countryService.getAllCountries();

        LOGGER.debug("countries={}", countries);

        LOGGER.info("End");

    }

* Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

        ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

        countryService = context.getBean(CountryService.class);

        testGetAllCountries();

* Execute main method to check if data from ormlearn database is retrieved.

**Hibernate XML Config implementation walk through**  
  
SME to provide explanation on the sample Hibernate implementation available in the link below:  
https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm  
  
Explanation Topics

* Explain how object to relational database mapping done in hibernate xml configuration file
* Explain about following aspects of implementing the end to end operations in Hibernate:
  + SessionFactory
  + Session
  + Transaction
  + beginTransaction()
  + commit()
  + rollback()
  + session.save()
  + session.createQuery().list()
  + session.get()
  + session.delete()

**Hibernate Annotation Config implementation walk through**  
  
SME to provide explanation on the sample Hibernate implementation available in the link below:  
https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm  
  
Explanation Topics

* Explain how object to relational database mapping done in persistence class file Employee
* Explain about following aspects of implementing the end to end operations in Hibernate:
  + @Entity
  + @Table
  + @Id
  + @GeneratedValue
  + @Column
  + Hibernate Configuration (hibernate.cfg.xml)
    - Dialect
    - Driver
    - Connection URL
    - Username
    - Password

**Difference between JPA, Hibernate and Spring Data JPA**  
  
Java Persistence API (JPA)

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

Hibernate

* ORM Tool that implements JPA

Spring Data JPA

* Does not have JPA implementation, but reduces boiler plate code
* This is another level of abstraction over JPA implementation provider like Hibernate
* Manages transactions

**Refer code snippets below on how the code compares between Hibernate and Spring Data JPA  
Hibernate**

   /\* Method to CREATE an employee in the database \*/

   public Integer addEmployee(Employee employee){

      Session session = factory.openSession();

      Transaction tx = null;

      Integer employeeID = null;

      try {

         tx = session.beginTransaction();

         employeeID = (Integer) session.save(employee);

         tx.commit();

      } catch (HibernateException e) {

         if (tx != null) tx.rollback();

         e.printStackTrace();

      } finally {

         session.close();

      }

      return employeeID;

   }

**Spring Data JPA**  
EmployeeRespository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowire

  private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

  employeeRepository.save(employee);

  }

​​​​​​​  
  
**Reference Links:**  
<https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1>  
<https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html>

**Implement services for managing Country**  
  
An application requires for features to be implemented with regards to country. These features needs to be supported by implementing them as service using Spring Data JPA.

* Find a country based on country code
* Add new country
* Update country
* Delete country
* Find list of countries matching a partial country name

Before starting the implementation of the above features, there are few configuration and data population that needs to be incorporated. Please refer each topic below and implement the same.  
  
**Explanation for Hibernate table creation configuration**

* Moreover the ddl-auto defines how hibernate behaves if a specific table or column is not present in the database.
  + create - drops existing tables data and structure, then creates new tables
  + validate - check if the table and columns exist or not, throws an exception if a matching table or column is not found
  + update - if a table does not exists, it creates a new table; if a column does not exists, it creates a new column
  + create-drop - creates the table, once all operations are completed, the table is dropped

# Hibernate ddl auto (create, create-drop, update, validate)

spring.jpa.hibernate.ddl-auto=validate

Populate country table

* Delete all the records in Country table and then use the below script to create the actual list of all countries in our world.

insert into country (co\_code, co\_name) values ("AF", "Afghanistan");

insert into country (co\_code, co\_name) values ("AL", "Albania");

insert into country (co\_code, co\_name) values ("DZ", "Algeria");

insert into country (co\_code, co\_name) values ("AS", "American Samoa");

insert into country (co\_code, co\_name) values ("AD", "Andorra");

insert into country (co\_code, co\_name) values ("AO", "Angola");

insert into country (co\_code, co\_name) values ("AI", "Anguilla");

insert into country (co\_code, co\_name) values ("AQ", "Antarctica");

insert into country (co\_code, co\_name) values ("AG", "Antigua and Barbuda");

insert into country (co\_code, co\_name) values ("AR", "Argentina");

insert into country (co\_code, co\_name) values ("AM", "Armenia");

insert into country (co\_code, co\_name) values ("AW", "Aruba");

insert into country (co\_code, co\_name) values ("AU", "Australia");

insert into country (co\_code, co\_name) values ("AT", "Austria");

insert into country (co\_code, co\_name) values ("AZ", "Azerbaijan");

insert into country (co\_code, co\_name) values ("BS", "Bahamas");

insert into country (co\_code, co\_name) values ("BH", "Bahrain");

insert into country (co\_code, co\_name) values ("BD", "Bangladesh");

insert into country (co\_code, co\_name) values ("BB", "Barbados");

insert into country (co\_code, co\_name) values ("BY", "Belarus");

insert into country (co\_code, co\_name) values ("BE", "Belgium");

insert into country (co\_code, co\_name) values ("BZ", "Belize");

insert into country (co\_code, co\_name) values ("BJ", "Benin");

insert into country (co\_code, co\_name) values ("BM", "Bermuda");

insert into country (co\_code, co\_name) values ("BT", "Bhutan");

insert into country (co\_code, co\_name) values ("BO", "Bolivia, Plurinational State of");

insert into country (co\_code, co\_name) values ("BQ", "Bonaire, Sint Eustatius and Saba");

insert into country (co\_code, co\_name) values ("BA", "Bosnia and Herzegovina");

insert into country (co\_code, co\_name) values ("BW", "Botswana");

insert into country (co\_code, co\_name) values ("BV", "Bouvet Island");

insert into country (co\_code, co\_name) values ("BR", "Brazil");

insert into country (co\_code, co\_name) values ("IO", "British Indian Ocean Territory");

insert into country (co\_code, co\_name) values ("BN", "Brunei Darussalam");

insert into country (co\_code, co\_name) values ("BG", "Bulgaria");

insert into country (co\_code, co\_name) values ("BF", "Burkina Faso");

insert into country (co\_code, co\_name) values ("BI", "Burundi");

insert into country (co\_code, co\_name) values ("KH", "Cambodia");

insert into country (co\_code, co\_name) values ("CM", "Cameroon");

insert into country (co\_code, co\_name) values ("CA", "Canada");

insert into country (co\_code, co\_name) values ("CV", "Cape Verde");

insert into country (co\_code, co\_name) values ("KY", "Cayman Islands");

insert into country (co\_code, co\_name) values ("CF", "Central African Republic");

insert into country (co\_code, co\_name) values ("TD", "Chad");

insert into country (co\_code, co\_name) values ("CL", "Chile");

insert into country (co\_code, co\_name) values ("CN", "China");

insert into country (co\_code, co\_name) values ("CX", "Christmas Island");

insert into country (co\_code, co\_name) values ("CC", "Cocos (Keeling) Islands");

insert into country (co\_code, co\_name) values ("CO", "Colombia");

insert into country (co\_code, co\_name) values ("KM", "Comoros");

insert into country (co\_code, co\_name) values ("CG", "Congo");

insert into country (co\_code, co\_name) values ("CD", "Congo, the Democratic Republic of the");

insert into country (co\_code, co\_name) values ("CK", "Cook Islands");

insert into country (co\_code, co\_name) values ("CR", "Costa Rica");

insert into country (co\_code, co\_name) values ("HR", "Croatia");

insert into country (co\_code, co\_name) values ("CU", "Cuba");

insert into country (co\_code, co\_name) values ("CW", "Curaçao");

insert into country (co\_code, co\_name) values ("CY", "Cyprus");

insert into country (co\_code, co\_name) values ("CZ", "Czech Republic");

insert into country (co\_code, co\_name) values ("CI", "Côte d'Ivoire");

insert into country (co\_code, co\_name) values ("DK", "Denmark");

insert into country (co\_code, co\_name) values ("DJ", "Djibouti");

insert into country (co\_code, co\_name) values ("DM", "Dominica");

insert into country (co\_code, co\_name) values ("DO", "Dominican Republic");

insert into country (co\_code, co\_name) values ("EC", "Ecuador");

insert into country (co\_code, co\_name) values ("EG", "Egypt");

insert into country (co\_code, co\_name) values ("SV", "El Salvador");

insert into country (co\_code, co\_name) values ("GQ", "Equatorial Guinea");

insert into country (co\_code, co\_name) values ("ER", "Eritrea");

insert into country (co\_code, co\_name) values ("EE", "Estonia");

insert into country (co\_code, co\_name) values ("ET", "Ethiopia");

insert into country (co\_code, co\_name) values ("FK", "Falkland Islands (Malvinas)");

insert into country (co\_code, co\_name) values ("FO", "Faroe Islands");

insert into country (co\_code, co\_name) values ("FJ", "Fiji");

insert into country (co\_code, co\_name) values ("FI", "Finland");

insert into country (co\_code, co\_name) values ("FR", "France");

insert into country (co\_code, co\_name) values ("GF", "French Guiana");

insert into country (co\_code, co\_name) values ("PF", "French Polynesia");

insert into country (co\_code, co\_name) values ("TF", "French Southern Territories");

insert into country (co\_code, co\_name) values ("GA", "Gabon");

insert into country (co\_code, co\_name) values ("GM", "Gambia");

insert into country (co\_code, co\_name) values ("GE", "Georgia");

insert into country (co\_code, co\_name) values ("DE", "Germany");

insert into country (co\_code, co\_name) values ("GH", "Ghana");

insert into country (co\_code, co\_name) values ("GI", "Gibraltar");

insert into country (co\_code, co\_name) values ("GR", "Greece");

insert into country (co\_code, co\_name) values ("GL", "Greenland");

insert into country (co\_code, co\_name) values ("GD", "Grenada");

insert into country (co\_code, co\_name) values ("GP", "Guadeloupe");

insert into country (co\_code, co\_name) values ("GU", "Guam");

insert into country (co\_code, co\_name) values ("GT", "Guatemala");

insert into country (co\_code, co\_name) values ("GG", "Guernsey");

insert into country (co\_code, co\_name) values ("GN", "Guinea");

insert into country (co\_code, co\_name) values ("GW", "Guinea-Bissau");

insert into country (co\_code, co\_name) values ("GY", "Guyana");

insert into country (co\_code, co\_name) values ("HT", "Haiti");

insert into country (co\_code, co\_name) values ("HM", "Heard Island and McDonald Islands");

insert into country (co\_code, co\_name) values ("VA", "Holy See (Vatican City State)");

insert into country (co\_code, co\_name) values ("HN", "Honduras");

insert into country (co\_code, co\_name) values ("HK", "Hong Kong");

insert into country (co\_code, co\_name) values ("HU", "Hungary");

insert into country (co\_code, co\_name) values ("IS", "Iceland");

insert into country (co\_code, co\_name) values ("IN", "India");

insert into country (co\_code, co\_name) values ("ID", "Indonesia");

insert into country (co\_code, co\_name) values ("IR", "Iran, Islamic Republic of");

insert into country (co\_code, co\_name) values ("IQ", "Iraq");

insert into country (co\_code, co\_name) values ("IE", "Ireland");

insert into country (co\_code, co\_name) values ("IM", "Isle of Man");

insert into country (co\_code, co\_name) values ("IL", "Israel");

insert into country (co\_code, co\_name) values ("IT", "Italy");

insert into country (co\_code, co\_name) values ("JM", "Jamaica");

insert into country (co\_code, co\_name) values ("JP", "Japan");

insert into country (co\_code, co\_name) values ("JE", "Jersey");

insert into country (co\_code, co\_name) values ("JO", "Jordan");

insert into country (co\_code, co\_name) values ("KZ", "Kazakhstan");

insert into country (co\_code, co\_name) values ("KE", "Kenya");

insert into country (co\_code, co\_name) values ("KI", "Kiribati");

insert into country (co\_code, co\_name) values ("KP", "Democratic People's Republic of Korea");

insert into country (co\_code, co\_name) values ("KR", "Republic of Korea");

insert into country (co\_code, co\_name) values ("KW", "Kuwait");

insert into country (co\_code, co\_name) values ("KG", "Kyrgyzstan");

insert into country (co\_code, co\_name) values ("LA", "Lao People's Democratic Republic");

insert into country (co\_code, co\_name) values ("LV", "Latvia");

insert into country (co\_code, co\_name) values ("LB", "Lebanon");

insert into country (co\_code, co\_name) values ("LS", "Lesotho");

insert into country (co\_code, co\_name) values ("LR", "Liberia");

insert into country (co\_code, co\_name) values ("LY", "Libya");

insert into country (co\_code, co\_name) values ("LI", "Liechtenstein");

insert into country (co\_code, co\_name) values ("LT", "Lithuania");

insert into country (co\_code, co\_name) values ("LU", "Luxembourg");

insert into country (co\_code, co\_name) values ("MO", "Macao");

insert into country (co\_code, co\_name) values ("MK", "Macedonia, the Former Yugoslav Republic of");

insert into country (co\_code, co\_name) values ("MG", "Madagascar");

insert into country (co\_code, co\_name) values ("MW", "Malawi");

insert into country (co\_code, co\_name) values ("MY", "Malaysia");

insert into country (co\_code, co\_name) values ("MV", "Maldives");

insert into country (co\_code, co\_name) values ("ML", "Mali");

insert into country (co\_code, co\_name) values ("MT", "Malta");

insert into country (co\_code, co\_name) values ("MH", "Marshall Islands");

insert into country (co\_code, co\_name) values ("MQ", "Martinique");

insert into country (co\_code, co\_name) values ("MR", "Mauritania");

insert into country (co\_code, co\_name) values ("MU", "Mauritius");

insert into country (co\_code, co\_name) values ("YT", "Mayotte");

insert into country (co\_code, co\_name) values ("MX", "Mexico");

insert into country (co\_code, co\_name) values ("FM", "Micronesia, Federated States of");

insert into country (co\_code, co\_name) values ("MD", "Moldova, Republic of");

insert into country (co\_code, co\_name) values ("MC", "Monaco");

insert into country (co\_code, co\_name) values ("MN", "Mongolia");

insert into country (co\_code, co\_name) values ("ME", "Montenegro");

insert into country (co\_code, co\_name) values ("MS", "Montserrat");

insert into country (co\_code, co\_name) values ("MA", "Morocco");

insert into country (co\_code, co\_name) values ("MZ", "Mozambique");

insert into country (co\_code, co\_name) values ("MM", "Myanmar");

insert into country (co\_code, co\_name) values ("NA", "Namibia");

insert into country (co\_code, co\_name) values ("NR", "Nauru");

insert into country (co\_code, co\_name) values ("NP", "Nepal");

insert into country (co\_code, co\_name) values ("NL", "Netherlands");

insert into country (co\_code, co\_name) values ("NC", "New Caledonia");

insert into country (co\_code, co\_name) values ("NZ", "New Zealand");

insert into country (co\_code, co\_name) values ("NI", "Nicaragua");

insert into country (co\_code, co\_name) values ("NE", "Niger");

insert into country (co\_code, co\_name) values ("NG", "Nigeria");

insert into country (co\_code, co\_name) values ("NU", "Niue");

insert into country (co\_code, co\_name) values ("NF", "Norfolk Island");

insert into country (co\_code, co\_name) values ("MP", "Northern Mariana Islands");

insert into country (co\_code, co\_name) values ("NO", "Norway");

insert into country (co\_code, co\_name) values ("OM", "Oman");

insert into country (co\_code, co\_name) values ("PK", "Pakistan");

insert into country (co\_code, co\_name) values ("PW", "Palau");

insert into country (co\_code, co\_name) values ("PS", "Palestine, State of");

insert into country (co\_code, co\_name) values ("PA", "Panama");

insert into country (co\_code, co\_name) values ("PG", "Papua New Guinea");

insert into country (co\_code, co\_name) values ("PY", "Paraguay");

insert into country (co\_code, co\_name) values ("PE", "Peru");

insert into country (co\_code, co\_name) values ("PH", "Philippines");

insert into country (co\_code, co\_name) values ("PN", "Pitcairn");

insert into country (co\_code, co\_name) values ("PL", "Poland");

insert into country (co\_code, co\_name) values ("PT", "Portugal");

insert into country (co\_code, co\_name) values ("PR", "Puerto Rico");

insert into country (co\_code, co\_name) values ("QA", "Qatar");

insert into country (co\_code, co\_name) values ("RO", "Romania");

insert into country (co\_code, co\_name) values ("RU", "Russian Federation");

insert into country (co\_code, co\_name) values ("RW", "Rwanda");

insert into country (co\_code, co\_name) values ("RE", "Réunion");

insert into country (co\_code, co\_name) values ("BL", "Saint Barthélemy");

insert into country (co\_code, co\_name) values ("SH", "Saint Helena, Ascension and Tristan da Cunha");

insert into country (co\_code, co\_name) values ("KN", "Saint Kitts and Nevis");

insert into country (co\_code, co\_name) values ("LC", "Saint Lucia");

insert into country (co\_code, co\_name) values ("MF", "Saint Martin (French part)");

insert into country (co\_code, co\_name) values ("PM", "Saint Pierre and Miquelon");

insert into country (co\_code, co\_name) values ("VC", "Saint Vincent and the Grenadines");

insert into country (co\_code, co\_name) values ("WS", "Samoa");

insert into country (co\_code, co\_name) values ("SM", "San Marino");

insert into country (co\_code, co\_name) values ("ST", "Sao Tome and Principe");

insert into country (co\_code, co\_name) values ("SA", "Saudi Arabia");

insert into country (co\_code, co\_name) values ("SN", "Senegal");

insert into country (co\_code, co\_name) values ("RS", "Serbia");

insert into country (co\_code, co\_name) values ("SC", "Seychelles");

insert into country (co\_code, co\_name) values ("SL", "Sierra Leone");

insert into country (co\_code, co\_name) values ("SG", "Singapore");

insert into country (co\_code, co\_name) values ("SX", "Sint Maarten (Dutch part)");

insert into country (co\_code, co\_name) values ("SK", "Slovakia");

insert into country (co\_code, co\_name) values ("SI", "Slovenia");

insert into country (co\_code, co\_name) values ("SB", "Solomon Islands");

insert into country (co\_code, co\_name) values ("SO", "Somalia");

insert into country (co\_code, co\_name) values ("ZA", "South Africa");

insert into country (co\_code, co\_name) values ("GS", "South Georgia and the South Sandwich Islands");

insert into country (co\_code, co\_name) values ("SS", "South Sudan");

insert into country (co\_code, co\_name) values ("ES", "Spain");

insert into country (co\_code, co\_name) values ("LK", "Sri Lanka");

insert into country (co\_code, co\_name) values ("SD", "Sudan");

insert into country (co\_code, co\_name) values ("SR", "Suriname");

insert into country (co\_code, co\_name) values ("SJ", "Svalbard and Jan Mayen");

insert into country (co\_code, co\_name) values ("SZ", "Swaziland");

insert into country (co\_code, co\_name) values ("SE", "Sweden");

insert into country (co\_code, co\_name) values ("CH", "Switzerland");

insert into country (co\_code, co\_name) values ("SY", "Syrian Arab Republic");

insert into country (co\_code, co\_name) values ("TW", "Taiwan, Province of China");

insert into country (co\_code, co\_name) values ("TJ", "Tajikistan");

insert into country (co\_code, co\_name) values ("TZ", "Tanzania, United Republic of");

insert into country (co\_code, co\_name) values ("TH", "Thailand");

insert into country (co\_code, co\_name) values ("TL", "Timor-Leste");

insert into country (co\_code, co\_name) values ("TG", "Togo");

insert into country (co\_code, co\_name) values ("TK", "Tokelau");

insert into country (co\_code, co\_name) values ("TO", "Tonga");

insert into country (co\_code, co\_name) values ("TT", "Trinidad and Tobago");

insert into country (co\_code, co\_name) values ("TN", "Tunisia");

insert into country (co\_code, co\_name) values ("TR", "Turkey");

insert into country (co\_code, co\_name) values ("TM", "Turkmenistan");

insert into country (co\_code, co\_name) values ("TC", "Turks and Caicos Islands");

insert into country (co\_code, co\_name) values ("TV", "Tuvalu");

insert into country (co\_code, co\_name) values ("UG", "Uganda");

insert into country (co\_code, co\_name) values ("UA", "Ukraine");

insert into country (co\_code, co\_name) values ("AE", "United Arab Emirates");

insert into country (co\_code, co\_name) values ("GB", "United Kingdom");

insert into country (co\_code, co\_name) values ("US", "United States");

insert into country (co\_code, co\_name) values ("UM", "United States Minor Outlying Islands");

insert into country (co\_code, co\_name) values ("UY", "Uruguay");

insert into country (co\_code, co\_name) values ("UZ", "Uzbekistan");

insert into country (co\_code, co\_name) values ("VU", "Vanuatu");

insert into country (co\_code, co\_name) values ("VE", "Venezuela, Bolivarian Republic of");

insert into country (co\_code, co\_name) values ("VN", "Viet Nam");

insert into country (co\_code, co\_name) values ("VG", "Virgin Islands, British");

insert into country (co\_code, co\_name) values ("VI", "Virgin Islands, U.S.");

insert into country (co\_code, co\_name) values ("WF", "Wallis and Futuna");

insert into country (co\_code, co\_name) values ("EH", "Western Sahara");

insert into country (co\_code, co\_name) values ("YE", "Yemen");

insert into country (co\_code, co\_name) values ("ZM", "Zambia");

insert into country (co\_code, co\_name) values ("ZW", "Zimbabwe");

insert into country (co\_code, co\_name) values ("AX", "Åland Islands");

Refer subsequent hands on exercises to implement the features related to country.

**Find a country based on country code**

* Create new exception class CountryNotFoundException in com.cognizant.spring-learn.service.exception
* Create new method findCountryByCode() in CountryService with @Transactional annotation
* In findCountryByCode() method, perform the following steps:
  + Method signature

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException

* Get the country based on findById() built in method

Optional<Country> result = countryRepository.findById(countryCode);

* From the result, check if a country is found. If not found, throw CountryNotFoundException

if (!result.isPresent())

* Use get() method to return the country fetched.

Country country = result.get();

* Include new test method in OrmLearnApplication to find a country based on country code and compare the country name to check if it is valid.

    private static void getAllCountriesTest() {

        LOGGER.info("Start");

        Country country = countryService.findCountryByCode("IN");

  LOGGER.debug("Country:{}", country);

        LOGGER.info("End");

    }

* Invoke the above method in main() method and test it.

**NOTE:** SME to explain the importance of @Transactional annotation. Spring takes care of creating the Hibernate session and manages the transactionality when executing the service method.

**Add a new country**

* Create new method in CountryService.

@Transactional

public void addCountry(Country country)

* Invoke save() method of repository to get the country added.

countryRepository.save(country)

* Include new testAddCountry() method in OrmLearnApplication. Perform steps below:
  + Create new instance of country with a new code and name
  + Call countryService.addCountry() passing the country created in the previous step.
  + Invoke countryService.findCountryByCode() passing the same code used when adding a new country
  + Check in the database if the country is added

**Update a country based on code**

* Create a new method updateCountry() in CountryService with parameters code and name. Annotate this method with @Transactional. Implement following steps in this method.
  + Get the reference of the country using findById() method in repository
  + In the country reference obtained, update the name of country using setter method
  + Call countryRepository.save() method to update the name
* Include new test method in OrmLearnApplication, which invokes updateCountry() method in CountryService passing a country's code and different name for the country.
* Check in database table if name is modified.

**Delete a country based on code**

* Create new method deleteCountry() in CountryService. Annotate this method with @Transactional.
* In deleteCountry() method call deleteById() method of repository.
* Include new test method in OrmLearnApplication with following steps
  + Call the delete method based on the country code during the add country hands on
* Check in database if the country is deleted

[Query Methods and O/R Mapping](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate implementation of Query Methods feature of Spring Data JPA
  + Query Methods - Search by containing text, sorting, filter with starting text, fetch between dates, greater than or lesser than, top
    - Query methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#jpa.query-methods.query-creation

* Demonstrate implementation of O/R Mapping
  + @ManyToOne, @JoinColumn, @OneToMany, FetchType.EAGER, FetchType.LAZY, @ManyToMany, @JoinTable, mappedBy
    - Relationships reference - https://www.baeldung.com/spring-data-rest-relationships

[](https://cognizant.e-box.co.in/projectComponent/show/15750)

**Write queries on country table using Query Methods**  
  
Following are the list of queries that is required for an application. Implement these queries using Query Methods feature of Spring Data JPA. Click [**here**](https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#jpa.query-methods.query-creation) for reference. Include appropriate methods in OrmLearnApplication and test the same.

* An application has a search text box for searching by country. When typing characters on the text box, a list of all the matching countries should be displayed. For example, if 'ou' is entered in the search box the following countries should be displayed. Write a Query Method to achieve this feature. Implement this method in CountryRepository.

BV       Bouvet Island

DJ       Djibouti

GP       Guadeloupe

GS       South Georgia and the South Sandwich Islands

LU       Luxembourg

SS       South Sudan

TF       French Southern Territories

UM       United States Minor Outlying Islands

ZA       South Africa

* Enhance the above method to return the countries in ascending order. Modify the query method name defined in the previous problem to achieve this.

BV       Bouvet Island

DJ       Djibouti

TF       French Southern Territories

GP       Guadeloupe

LU       Luxembourg

ZA       South Africa

GS       South Georgia and the South Sandwich Islands

SS       South Sudan

UM       United States Minor Outlying Islands

* To select a country an alphabet index is displayed in a web page, when the user clicks on the alphabet, all the countries starting that alphabet needs to be displayed. For example if the alphabet choose is 'Z', then the following countries should be displayed. Write a query method to get this feature incorporated.

ZM       Zambia

ZW       Zimbabwe

**Test Cases**

**Write queries on stock table using Query Methods**  
  
With one year stock data of Facebook, Google and Netflix, we need to implement Spring Data JPA Query Methods for the following scenarios:  
  
**Clone sample data folder from git:**  
Sample data for implementing this hands on is available in the git server. Follow steps below to download the same.

* Identify a convenient folder for saving the same data files
* In Windows File Explorer go to the folder identified
* Right click on the empty space of this folder and select 'Git Bash'
* Execute the following command which downloads all necessary files:

git clone https://code.cognizant.com/genc-fse-java/spring-data-jpa-files.git

**Setup stock data**

* Create a new table for storing stock details.

CREATE TABLE IF NOT EXISTS `ormlearn`.`stock` (

  `st\_id` INT NOT NULL AUTO\_INCREMENT,

  `st\_code` varchar(10),

  `st\_date` date,

  `st\_open` numeric(10,2),

  `st\_close` numeric(10,2),

  `st\_volume` numeric,

  PRIMARY KEY (`st\_id`)

);

* The file stock-data.csv in spring-data-jpa-files folder contains the stock data of Facebook, Google and Netflix from 18 Oct 2018 to 17 Oct 2019. This is public data downloaded from finance.yahoo.com.
* Open stock-data.csv file in Excel
* Include the following formula in F2 cell, this will display the insert script in F2 cell.

=CONCATENATE("insert into stock (st\_code, st\_date, st\_open, st\_close, st\_volume) values (""", B2, """, """, YEAR(A2), "-", MONTH(A2), "-", DAY(A2), """, ", C2, ", ", D2, ", ", E2, ");")

* Drag the formula for all the rows of data in F column.
* Copy all data in F column and paste it in Notepad or Notepad++ and save it as a file named stock-data.sql. Execute this script in mysql command line client which populates data into ormlearn.stock table. Following command assumes that the git project is available in D:.

mysql>source D:\spring-data-jpa-files\stock-data.sql

* Create new class Stock in orm-learn project and define the required mapping annotations.
* Create StockRepository class to write the Query Methods
* Create methods in OrmLearnApplication to test by autowiring StockRepository directly.

**Query Methods required for the following scenarios**

* Get all stock details of Facebook in the month of September 2019. Expected data result of Query Method below.

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

+---------+------------+---------+----------+-----------+

| FB      | 2019-09-03 |  184.00 |   182.39 |   9779400 |

| FB      | 2019-09-04 |  184.65 |   187.14 |  11308000 |

| FB      | 2019-09-05 |  188.53 |   190.90 |  13876700 |

| FB      | 2019-09-06 |  190.21 |   187.49 |  15226800 |

| FB      | 2019-09-09 |  187.73 |   188.76 |  14722400 |

| FB      | 2019-09-10 |  187.44 |   186.17 |  15455900 |

| FB      | 2019-09-11 |  186.46 |   188.49 |  11761700 |

| FB      | 2019-09-12 |  189.86 |   187.47 |  11419800 |

| FB      | 2019-09-13 |  187.33 |   187.19 |  11441100 |

| FB      | 2019-09-16 |  186.93 |   186.22 |   8444800 |

| FB      | 2019-09-17 |  186.66 |   188.08 |   9671100 |

| FB      | 2019-09-18 |  188.09 |   188.14 |   9681900 |

| FB      | 2019-09-19 |  188.66 |   190.14 |  10392700 |

| FB      | 2019-09-20 |  190.66 |   189.93 |  19934200 |

| FB      | 2019-09-23 |  189.34 |   186.82 |  13327600 |

| FB      | 2019-09-24 |  187.98 |   181.28 |  18546600 |

| FB      | 2019-09-25 |  181.45 |   182.80 |  18068300 |

| FB      | 2019-09-26 |  181.33 |   180.11 |  16083300 |

| FB      | 2019-09-27 |  180.49 |   177.10 |  14656200 |

+---------+------------+---------+----------+-----------+

* Get all google stock details where the stock price was greater than 1250

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

+---------+------------+---------+----------+-----------+

| GOOGL   | 2019-04-22 | 1236.67 |  1253.76 |    954200 |

| GOOGL   | 2019-04-23 | 1256.64 |  1270.59 |   1593400 |

| GOOGL   | 2019-04-24 | 1270.59 |  1260.05 |   1169800 |

| GOOGL   | 2019-04-25 | 1270.30 |  1267.34 |   1567200 |

| GOOGL   | 2019-04-26 | 1273.38 |  1277.42 |   1361400 |

| GOOGL   | 2019-04-29 | 1280.51 |  1296.20 |   3618400 |

| GOOGL   | 2019-10-17 | 1251.40 |  1252.80 |   1047900 |

+---------+------------+---------+----------+-----------+

* Find the top 3 dates which had highest volume of transactions

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

+---------+------------+---------+----------+-----------+

| FB      | 2019-01-31 |  165.60 |   166.69 |  77233600 |

| FB      | 2018-10-31 |  155.00 |   151.79 |  60101300 |

| FB      | 2018-12-19 |  141.21 |   133.24 |  57404900 |

+---------+------------+---------+----------+-----------+

* Identify three dates when Netflix stocks were the lowest

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

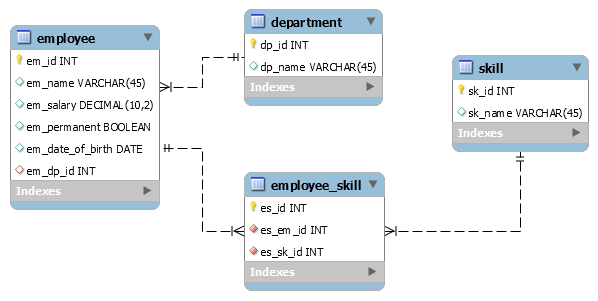
+---------+------------+---------+----------+-----------+

| NFLX    | 2018-12-24 |  242.00 |   233.88 |   9547600 |

| NFLX    | 2018-12-21 |  263.83 |   246.39 |  21397600 |

| NFLX    | 2018-12-26 |  233.92 |   253.67 |  14402700 |

+---------+------------+---------+----------+-----------+

**Create payroll tables and bean mapping**  
  
To demonstrate one to many, many to one and many to many relationships in Hibernate, a schema with entities employee, department and skill will be used. In this hands on we will setup the tables and data, which forms the basis for learning the mappings in Hibernate.  
  
**Schema Structure**  
  
  
  
Follow steps below to create necessary tables:

* Open mysql client in command line
* Use the source command to execute the payroll.sql script file available in spring-data-jpa-files folder. The following command assumes that spring-data-jpa-files folder is in D:.

mysql> source D:\spring-data-jpa-files\stock-data.sql

Define bean mapping

* Open orm-learn project in Eclipse
* Create model classes Employee, Department and Skill in com.cognizant.orm-learn.model package
* Define each model should have @Entity and @Table annotations.
* Each id field should be have @Id annotation and @GeneratedValue(strategy = GenerationType.IDENTITY) annotation. @GeneratedValue annotation ensures auto increment of id creation.
* Define @Column against each field.
* Define getters, setters and toString() methods
* Employee
  + private int id;
  + private String name;
  + private double salary;
  + private boolean permanent;
  + private Date dateOfBirth;
* Department
  + private int id;
  + private String name;
* Skill
  + private int id;
  + private String name;
* Create appropriate repository interfaces EmployeeRepository, DepartmentRepository and SkillRepository in repository package

**Implement many to one relationship between Employee and Department**  
  
Follow steps below to defined many to one relationship and perform persistence operations:  
  
**Preparation of Service Classes**

* Create EmployeeService, DepartmentService and SkillService defined with annotation @Service. In each of this class autowire respective repository.
* In each of the service class implement two methods one is to get the entity based on id and the other one is to save the entity. Sample code below provided for EmployeeService, in similar fashion include the methods for DepartmentService and SkillService.
* EmployeeService - get() method

    @Transactional

    public Employee get(int id) {

        LOGGER.info("Start");

        return employeeRepository.findById(id).get();

    }

* EmployeeService - save() method

    @Transactional

    public void save(Employee employee) {

        LOGGER.info("Start");

        employeeRepository.save(employee);

        LOGGER.info("End");

    }

* Include static references of EmployeeService, DepartmentService and SkillService in OrmLearnApplication.
* Assign employeeService, departmentService and skillService from the context in OrmLearnApplication main() method.

**Implementation of @ManyToOne mapping**

* Define department in Employee bean with @ManyToOne and @JoinTable annotation. This defines the relationship between the entities.

    @ManyToOne

    @JoinColumn(name = "em\_dp\_id")

    private Department department;

* Include setters and getters for instance variable department.

**Getting Employee along with Department**

* Create new method testGetEmployee() in OrmLearnApplication
* Implement below code in the method.

    private static void testGetEmployee() {

        LOGGER.info("Start");

        Employee employee = employeeService.get(1);

        LOGGER.debug("Employee:{}", employee);

        LOGGER.debug("Department:{}", employee.getDepartment());

        LOGGER.info("End");

    }

* The above implementation gets the employee with id 1 and displays the employee details and department details.
* Include testGetEmployee() method in main and comment the other test method calls.
* Execute the main method and observe the following:
  + In the logs check the lines where the query is generated.
  + Since the relationship is defined, hibernate fetches department data as well. The query should look something like the below. Observe the department table join in this query. The query is formatted for better readability.

select employee0\_.em\_id as em\_id1\_2\_0\_, employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_0\_, employee0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_,

  employee0\_.em\_name as em\_name3\_2\_0\_, employee0\_.em\_permanent as em\_perma4\_2\_0\_, employee0\_.em\_salary as em\_salar5\_2\_0\_,

  department1\_.dp\_id as dp\_id1\_1\_1\_, department1\_.dp\_name as dp\_name2\_1\_1\_

  from employee employee0\_ left outer join department department1\_

  on employee0\_.em\_dp\_id=department1\_.dp\_id

 where employee0\_.em\_id=?

**NOTE:** SME to explain the learners about Eager Fetch and Lazy Fetch. As per JPA specification by default, Eager Fetch is applied For ManyToOne and OneToOne relationships. Hence department details as well is joined and fetched by Hibernate.  
   
**Add Employee**

* Create new method testAddEmployee() in OrmLearnApplication and implement the following steps
  + Create a new instance of Employee
  + Set the values for the employee using setter method
  + Get a department based on department id 1 using departmentService
  + Set the department in employee based on the department obtained in the previous step
  + Invoke employeeService.save() passing the employee object created
  + Log employee object reference in debug mode
* Include testAddEmployee() invocation in main() method and comment other test methods
* Invoke the main method and check the following:
  + Log should contain select query to get the department and insert statement to add employee
  + Observe that the employee log after save contains the id. Hibernate inserts the records, gets the id and set the id instance variable of employee
  + Check in database if new employee data is inserted in employee table

**Update Employee**

* Create new method testUpdateEmployee() in OrmLearnApplication and implement the following steps
  + Get an employee instance based on employee id using employeeService.get() method
  + Get a department based on department id using departmentService. Use a different department id from the one that is fetched.
  + Set the department in employee based on the department obtained in the previous step
  + Invoke employeeService.save() passing the employee object created
  + Log employee object reference in debug mode
* Include testUpdateEmployee() invocation in main() method and comment other test methods
* Invoke the main method and check the following:
  + Log should contain select query to get the department and employee. There should be update statement that updates employee table
  + Check in database if department id is modified.

**Implement one to many relationship between Employee and Department**  
  
Department.java

* Include new instance variable for set of employees and define the OneToMay annotation

    @OneToMany(mappedBy = "department")

    private Set<Employee> employeeList;

* Include setter and getter for employeeList

OrmLearnApplication.java

* Include new method testGetDepartment()
* In this method, get a department using departmentService.get() passing the id. Select an department id that has more than one employee.
* Log the returned department and department.getEmployeeList()
* Include testGetDepartment() method invocation in main method and comment the other test methods.
* Execute the main() method which will fail with LazyInitializationException. This is because the default fetch type for OneToMany relationship is LAZY, hibernate fetches only department details and does not get the employee details.
* In order to get the employee list as well, modify the annotation to include the fetch type as EAGER. Make this change in employeeList annotation definition of Department class.

    @OneToMany(mappedBy = "department", fetch = FetchType.EAGER)

    private Set<Employee> employeeList;

* After this change try executing the main() method, which will fetch both department and employee

**Implement many to many relationship between Employee and Skill**  
  
**Many to Many mapping defintion**

* Include set of skill list in Employee.java with appropriate getter and setter
* Include set of employee list in Skill.java with appropriate getter and setter
* Include ManyToMany definition in Employee.java as specified below:

    @ManyToMany

    @JoinTable(name = "employee\_skill",

        joinColumns = @JoinColumn(name = "es\_em\_id"),

        inverseJoinColumns = @JoinColumn(name = "es\_sk\_id"))

    private Set<Skill> skillList;

* Include ManyToMany defintion in Skill.java as specified below:

    @ManyToMany(mappedBy = "skillList")

    private Set<Employee> employeeList;

**Fetching Employee along with Skills**

* In testGetEmployee() method of OrmLearnApplication.java include a new line to log employee skill details after the department log.

        LOGGER.debug("Skills:{}", employee.getSkillList());

* Execute the main method which will fail with LazyInitializationException
* Include fetch type as eager in @ManyToMany annotation of Employee.java, which will fetch the skill details as well.

**Add Skill to Employee**

* Include a new method testAddSkillToEmployee() in OrmLearnApplication.java
* Implement the following steps in this method:
  + Identify an employee id and skill id for which a relationship does not exists
  + Get employee based on employee id calling employeeService.get() method
  + Get skill based on skill id calling skillService.get() method
  + Get the skill list from employee and add the skill obtained in the previous step to the skill list
  + Call save method in employeeService passing the employee reference
* Invoke testAddSkillToEmployee() in main method and comment other test methods
* Execute the main method and check employee\_skill table to verify if the skill is added to the employee.

[Hibernate Query Language (HQL)](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate writing Hibernate Query Language and Native Query
  + HQL stands for Hibernate Query Language, JPQL stands for Java Persistence Query Language, Compare HQL and JPQL, @Query annotation, HQL fetch keyword, aggregate functions in HQL, Native Query, nativeQuery attribute
    - Reference - https://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html/ch11.html
    - Features of JPA Query - https://www.baeldung.com/spring-data-jpa-query

* Explain the need and benefit of Criteria Query
  + Scenarios where Criteria Query helps, CriteriaBuilder, Criteria Query, Root, TypedQuery
    - Reference - https://docs.oracle.com/javaee/6/tutorial/doc/gjrij.html

**Introduction to HQL and JPQL**

* HQL stands for Hibernate Query Language
* JPQL stands for Java Persistence Query Language
* Both HQL and JPQL are object focused query language similar to SQL
* JPQL is a subset of HQL
* All JPQL queries are valid HQL query, but the reverse is not true
* Both HQL and JPQL allows SELECT, UPDATE and DELETE
* HQL additionally allows INSERT statement

Reference - <https://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html/ch11.html>

**Get all permanent employees using HQL**  
  
Using HQL get all permanent employees. When retrieving the employee details it should also retrieve respective department and skill list as well.  
  
**HQL Solution**

* Include a new method definition in EmployeeRepository with @Query annotation

    @Query(value="SELECT e FROM Employee e WHERE e.permanent = 1")

    List<Employee> getAllPermanentEmployees();

  // NOTE: HQL looks like SQL, instead of table, Java classes and it's

  // instance variables are addressed here

* Include appropriate service method
* Include a new test method and Invoke the service method in OrmLearnApplication.java. Refer test method below that logs all employee details and each employee's skill details.

    public static void testGetAllPermanentEmployees() {

        LOGGER.info("Start");

        List<Employee> employees = employeeService.getAllPermanentEmployees();

        LOGGER.debug("Permanent Employees:{}", employees);

        employees.forEach(e -> LOGGER.debug("Skills:{}", e.getSkillList()));

        LOGGER.info("End");

    }

* Check the list of SQL queries executed in the log file. Following queries would have got executed.

select employee0\_.em\_id as em\_id1\_2\_0\_, department1\_.dp\_id as dp\_id1\_1\_1\_, skill3\_.sk\_id as sk\_id1\_4\_2\_, employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_0\_,

employee0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employee0\_.em\_name as em\_name3\_2\_0\_, employee0\_.em\_permanent as em\_perma4\_2\_0\_,

  employee0\_.em\_salary as em\_salar5\_2\_0\_, department1\_.dp\_name as dp\_name2\_1\_1\_, skill3\_.sk\_name as sk\_name2\_4\_2\_,

  skilllist2\_.es\_em\_id as es\_em\_id1\_3\_0\_\_, skilllist2\_.es\_sk\_id as es\_sk\_id2\_3\_0\_\_

from employee employee0\_ left outer join department department1\_ on

employee0\_.em\_dp\_id=department1\_.dp\_id left outer join employee\_skill skilllist2\_ on

employee0\_.em\_id=skilllist2\_.es\_em\_id left outer join skill skill3\_ on

skilllist2\_.es\_sk\_id=skill3\_.sk\_id

where employee0\_.em\_permanent=1

select employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employeeli0\_.em\_id as em\_id1\_2\_0\_, employeeli0\_.em\_id as em\_id1\_2\_1\_,

  employeeli0\_.em\_date\_of\_birth as em\_date\_2\_2\_1\_, employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_1\_,

  employeeli0\_.em\_name as em\_name3\_2\_1\_, employeeli0\_.em\_permanent as em\_perma4\_2\_1\_, employeeli0\_.em\_salary as em\_salar5\_2\_1\_

from employee employeeli0\_ where employeeli0\_.em\_dp\_id=3

select employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employeeli0\_.em\_id as em\_id1\_2\_0\_, employeeli0\_.em\_id as em\_id1\_2\_1\_,

  employeeli0\_.em\_date\_of\_birth as em\_date\_2\_2\_1\_, employeeli0\_.em\_dp\_id as em\_dp\_id6\_2\_1\_,

  employeeli0\_.em\_name as em\_name3\_2\_1\_, employeeli0\_.em\_permanent as em\_perma4\_2\_1\_, employeeli0\_.em\_salary as em\_salar5\_2\_1\_

from employee employeeli0\_ where employeeli0\_.em\_dp\_id=2

select skilllist0\_.es\_em\_id as es\_em\_id1\_3\_0\_, skilllist0\_.es\_sk\_id as es\_sk\_id2\_3\_0\_, skill1\_.sk\_id as sk\_id1\_4\_1\_, skill1\_.sk\_name as sk\_name2\_4\_1\_

from employee\_skill skilllist0\_ inner join skill skill1\_ on skilllist0\_.es\_sk\_id=skill1\_.sk\_id

where skilllist0\_.es\_em\_id=2

**Optimizing HQL Solution by removing the EAGER fetch configuration**

* An optimal solution should not execute multiple queries, we have defined unnecessary fetch configuration in
* Eager fetch configuration is defined for employeeList in Department.java and skillList of Employee.java
* Remove these two eager fetch configurations and check the logs. The following queries would have got executed. It would have failed when getting the skill list. Since we have remove the eager fetch skill is not retrieved.

select employee0\_.em\_id as em\_id1\_2\_, employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_, employee0\_.em\_dp\_id as em\_dp\_id6\_2\_, employee0\_.em\_name as em\_name3\_2\_,

  employee0\_.em\_permanent as em\_perma4\_2\_, employee0\_.em\_salary as em\_salar5\_2\_

from employee employee0\_

where employee0\_.em\_permanent=1

select department0\_.dp\_id as dp\_id1\_1\_0\_, department0\_.dp\_name as dp\_name2\_1\_0\_

from department department0\_

where department0\_.dp\_id=2

select department0\_.dp\_id as dp\_id1\_1\_0\_, department0\_.dp\_name as dp\_name2\_1\_0\_

from department department0\_

where department0\_.dp\_id=3

* There are two issues in this approach:
  + We did not get the skill details
  + Still the query is not optimal as we have three queries executed

**Optimizing HQL solution by using 'fetch'**

* Change the query in EmployeeRepository.java as specified below:

@Query(value="SELECT e FROM Employee e left join e.department d left join e.skillList WHERE e.permanent = 1")

* The above query still fails to get skill details. Include fetch after each join. Wherever data is required we can include fetch, which will populate the respective data. Change the query as specified below:

@Query(value="SELECT e FROM Employee e left join fetch e.department d left join fetch e.skillList WHERE e.permanent = 1")

* Following the single query generated for the above HQL:

select employee0\_.em\_id as em\_id1\_2\_0\_, department1\_.dp\_id as dp\_id1\_1\_1\_, skill3\_.sk\_id as sk\_id1\_4\_2\_,

  employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_0\_, employee0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employee0\_.em\_name as em\_name3\_2\_0\_,

  employee0\_.em\_permanent as em\_perma4\_2\_0\_, employee0\_.em\_salary as em\_salar5\_2\_0\_, department1\_.dp\_name as dp\_name2\_1\_1\_,

  skill3\_.sk\_name as sk\_name2\_4\_2\_, skilllist2\_.es\_em\_id as es\_em\_id1\_3\_0\_\_, skilllist2\_.es\_sk\_id as es\_sk\_id2\_3\_0\_\_

from employee employee0\_ left outer join department department1\_ on

  employee0\_.em\_dp\_id=department1\_.dp\_id left outer join employee\_skill skilllist2\_ on

  employee0\_.em\_id=skilllist2\_.es\_em\_id left outer join skill skill3\_ on

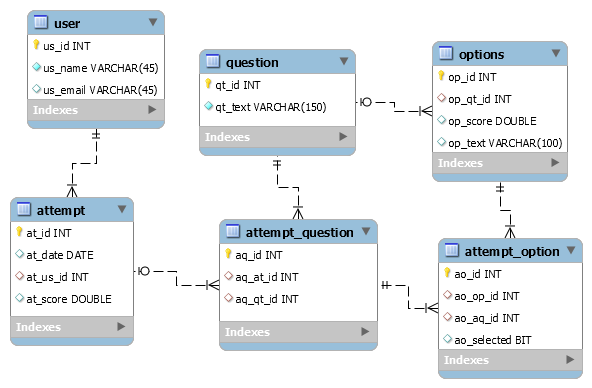
  skilllist2\_.es\_sk\_id=skill3\_.sk\_id

where employee0\_.em\_permanent=1

**IMPORTANT TAKEAWAY:** Join keyword links the table, but does not populate the beans. Fetch ensures that the beans are populated. Based on our need wherever we need data, we can define fetch. When joining table data is not needed the fetch can be ignored.

**Fetch quiz attempt details using HQL**  
  
In a quiz application there is a requirement for admin to view details of a quiz that an user had attempted. This view should include the following details:

* Username
* Attempted Date
* All questions as part of the attempt
* List of options under each quiz
* The option that is correct answer
* The score for correct answer

**Schema Diagram**  
  
Notes on Schema:

* Tables user, question and options are self explanatory. They hold the respective master data.
* Tables attempt, attempt\_question and attempt\_option are used to hold the data of attempts made by each user.

**Follow steps below to setup the schema:**

* Go to spring-data-jpa-files folder in windows explorer
* Open file quiz.mwb in MySQL Workbench
* Generate SQL file using File > Export > Forward Engineer SQL CREATE Script
* Click Browse and select the file name and folder for the saving the generated SQL file
* Select the check box "Generate INSERT Statements for Tables"
* Click Next > Next > Finish to generate the SQL file
* Execute the SQL file in ormlearn schema and check the data in the tables

**Steps to get this implemented:**

* Create necessary entity class for each table defined above
* Define necessary O/R mapping based on the schema defined above​​​
* Create a Repository and Service class:
  + AttemptRepository
    - public Attempt getAttempt(int userId, int attemptId)
  + AttemptService
    - public Attempt getAttempt(int userId, int attemptId)
* Modify OrmLearnApplication.java to include a new test method and test AttemptService.getAttemptDetail() method
* Create HQL that joins the tables in the below order:
  + user
  + attempt
  + attempt\_question
  + question
  + attempt\_option
  + options
* In the HQL include where class for userId and attemptId
* Include 'fetch' in HQL wherever there is one-to-many or many-to-many relationship
* In OrmLearnApplication.java test method ​​​​​​get the attempts details, iterate through the details and display the data in the following format. The second column in each option denotes the score from question table. The last column in each option denotes the answer selected by the user.

What is the extension of the hyper text markup language file?

1) .xhtm       0.0     false

2) .ht         0.0     false

3) .html       1.0     true

4) .htmx       0.0     false

What is the  maximum level of heading tag can be used in a HTML page?

1) 5           0.0   false

2) 3           0.0   true

3) 4         0.0   false

4) 6   1.0   false

The HTML document itself begins with <html> and  ends </html>. State True of False

1) false        0.0   false

2) true       1.0   true

Choose the right option to store text value value in a variable

1) 'John'     0.5    true

2) John       0.0    false

3) "John" 0.5    false

4) /John/ 0.0    false

**Get average salary using HQL**  
  
Compute the average salary of a department using HQL.  
  
Refer steps below to implement:

* Define HQL in EmployeeRepository

    @Query(value="SELECT AVG(e.salary) FROM Employee e")

    double getAverageSalary();

* Include new method with above signature in EmployeeService and include test method in OrmLearnApplication
* The above query does not filter the result based on department id. Modify the query and method signature as specified below to accept department.

    @Query(value="SELECT AVG(e.salary) FROM Employee e where e.department.id = :id")

    double getAverageSalary(@Param("id") int id);

**NOTES:**

* Observe how department id is referred from 'e'
* Make note of the colon (:) used to define a parameter within a query
* @Param annotation helps in binding the input department id with the query parameter
* Similar to AVG(), all other aggregate functions can be used

**Get all employees using Native Query**  
  
About Native Queries

* Native queries are direct SQL queries to the database instead of using HQL
* Try to avoid Native Queries and make it minimal.
* Avoiding native queries helps in easier portability of database

Follow steps below to implement

* Define a new native query method in EmployeeRepository

    @Query(value="SELECT \* FROM employee", nativeQuery = true)

    List<Employee> getAllEmployeesNative();

* Define relevant method in service and OrmLearnApplication and test it

**Criteria Query**  
  
Find below a online retail user scenario

* User goes to Amazon
* Searches with keyword "laptop"
* The left hand size contains the following filteria criteria categories:
  + Customer review
  + Hard Disk Size
  + RAM Size
  + CPU Speed
  + Operating System
  + Weight
  + CPU
* The user might select options available in one or more of the criteria and try a fresh search.

In the above given scenario, what will be the where clause of the HQL query that you will run on the product?  
  
The where clause varies based on the criteria selected by the user. We have to dynamically frame the where clause filters based on the criteria selected by user.  
  
Criteria Query helps in handling this scenario in a better way. The filter criteria can be programmatically added, rather than fixing the HQL Statement.  
  
Go through the examples in the link below to understand how Criteria Query has to be implemented.  
<https://howtodoinjava.com/hibernate/hibernate-criteria-queries-tutorial/>

[REST API Integration](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate implementation of RESTful Web Service integrated with Spring Data JPA
  + @JsonIgnore, @JsonManagedReference, @JsonBackReference, @JsonIdentityInfo, data transfer objects, persisting data in POST and PUT methods
    - Reference to resolve cyclic dependency - https://www.baeldung.com/jackson-bidirectional-relationships-and-infinite-recursion

**REST API to get all countries**  
  
Follow steps below to implement:

* The REST API will be implemented in the spring-learn project
* Modify spring-learn application pom.xml to include Spring Data JPA and MySQL Connector dependencies
* Modify application.properties of spring-learn application to include database connectivity details and hibernate log configuration details
* Create com.spring-learn.repository and com.spring-learn.service packages
* Copy CountryService and CountryRepository from orm-learn project
* Autowire CountryService in CountryController and modify the respective controller method that returns all countries to use CountryService
* Run the service
* Using Chrome Browser or curl command or postman to test the REST API and check if it returns the countries from the database

**Implement all REST API for Country**  
  
Using the approach defined in the previous hands on implement the following REST API services related to country:

* Get country by code
  + Endpoint - /countries/IN
  + Method - GET
  + Expected Output

{"code":"IN","name":"India"}

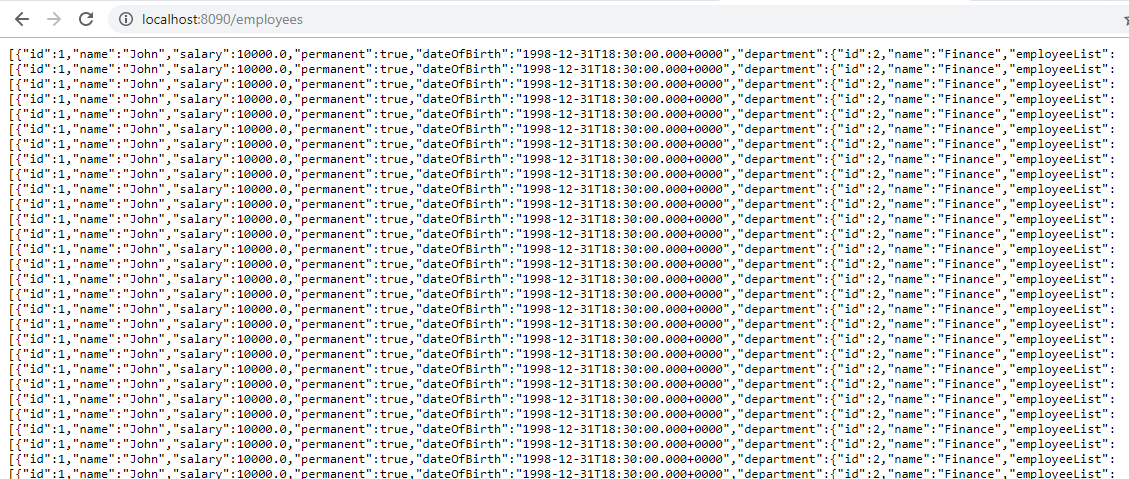
* Get country based on keyword with result sorted based on country name
  + Endpoint - /countries?key=ambia
  + Method - GET
  + Expected Output

[{"code":"GM","name":"Gambia"}, {"code":"ZM","name":"Zambia"}]

* Add Country
  + Endpoint - /countries
  + Method - POST
  + Request Body - New country details in JSON format {"code":"IN","name":"India"}
  + Validations
    - code is required
    - code cannot exceed 2 characters
    - name is required
    - name cannot exceed 50 characters
  + Response - HTTP Status code 200
* Update Country
  + Endpoint - /countries
  + Method - PUT
  + Request Body - New country details in JSON format {"code":"IN","name":"India"}
  + Same validations as specified for add country
  + Response - HTTP Status code 200
* Delete Country
  + Endpoint - /countries/IN
  + Method - DELETE
  + Response - HTTP Status code 200

**Implement get all employees as REST API**  
  
Similar to the instructions provided in previous hands on, create respective repository and service class in spring-learn application to implement the below REST API service.  
  
Get all employees

* Endpoint - /employees
* Method - GET
* Expected Result - Due to cyclic dependency of objects, it goes into a recursive loop and it does not result in the result that we expect. We will look at possible solutions in the next hands on.



**Test Cases**

**Resolutions of Cyclic Dependency to get all employees**  
  
**Solution using @JsonIgnore**  
  
Include @JsonIgnore annotation in the below given attributes:

* Department.employeeList
* Skill.employeeList

Note: Import JsonIgnore from com.fasterxml.jackson.annotation.  
  
The above changes will resolve the issue and data will be retrieved as expected. But there is a disadvantage to this approach. If there is a need to get all employees based on Skill, the employee list cannot be retrieved due to @JsonIgnore.  
  
**Solution using @JsonManagedReference and @JsonBackReference**  
  
Comment out the @JsonIgnore annotations defined earlier.  
  
Refer implementation available in the link below and check if this solves the cyclic dependency issue. There is possibility that few employee details are repeated in the result. Hence this might not bean ideal solution.  
<https://www.baeldung.com/jackson-bidirectional-relationships-and-infinite-recursion>  
  
**Solution using @JsonIdentityInfo**  
  
Comment out the @JsonManagedReference and @JsonBackReference annotations and try the @JsonIdentityInfo feature provided in the above link. This goes for a recursive data retrieval and will not be in the form that we need.  
  
**Solution using Data Transfer Object**  
  
A new set of beans for Employee, Department and Skill needs to be created with DTO suffixed. Then transfer the data from repository into these new DTO objects and return the EmployeeDTO reference instead of the entity Employee reference. Follow steps below to implement this approach:

* Create new package com.cognizant.spring-learn.dto
* Create three classes in this new package named EmployeeDTO, DepartmentDTO and SkillDTO
* In the new DTO classes, create attribute exactly the same as the ones in model package
* Create a private method in EmployeeController with signature as defined below

private EmployeeDTO[] transformEmployeeToDTO(Employee[] employee)

* In this method create new EmployeeDTO instance of each Employee. For each employee create necessary DepartmentDTO and SkillDTO instances. Call respective DTO's setter method to set the data.
* Invoke this method in getAllEmployees() method after getting the employee list from repository.
* Return EmployeeDTO reference returned by the above method as return type for EmployeeController.getAllEmployees() method.

**Persist employee details through REST API**  
  
Expose the following RESTful API Services related to employee in spring-learn project.  
  
**Update Employee**

* Endpoint - /employees
* Method - PUT
* Request Body - Employee details in JSON Format
* Response - Status 200
* Implementation Steps
  + Copy necessary repository and service classes
  + Implement the REST API in EmployeeController
  + Use the existing request availablein Postman to make the PUT call and test if employee details are getting save
  + Kindly note that DTO classes are not required for this implementation. The entity objects can be directly used.

**Add Employee**

* Endpoint - /employees
* Method - PUT
* Request Body - Employee details in JSON Format
* Response - Status 200

**Verify transaction feature when adding employee**  
  
In the add employee REST API implemented in the previous hands on change the department id in the request JSON with an id that does not exist in the database.  
  
This request will respond with 500 - Internal Server Error.  
  
Observe the following aspects in the log:

* Insert SQL for employee table would have got executed successfully
* After that it would have failed due to non-availability of department id
* If we go to the database and check employee table, the newly inserted record will not be existing
* Due to failure to update the department details, the insertion had got rolled back

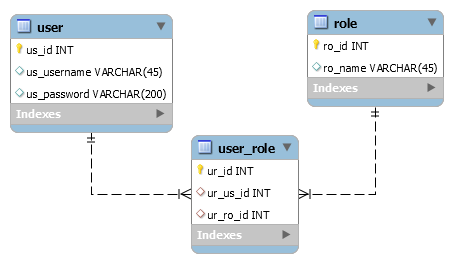
[Angular Integration and Mockito](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate implementation of spring security integrated with repository to get user details
  + UserDetailsService, UserDetail, loadUserByUsername(), UsernameNotFoundException
    - Reference - https://docs.spring.io/spring-security/site/docs/5.2.1.BUILD-SNAPSHOT/reference/htmlsingle/#tech-userdetailsservice
* Demonstrate implementation of isolated testing of service using Mockito
  + Importance of isolated testing, MockitoJUnitRunner, Mockito.mock(), when(), thenReturn()
    - Spring Boot with Mockito - https://www.springboottutorial.com/spring-boot-unit-testing-and-mocking-with-mockito-and-junit
* Demonstrate code coverage report generation on unit test cases executed
  + Eclipse, ECL Emma, understanding the color coding
    - EMMA Reference - https://www.eclemma.org/
* Demonstrate end to end integration with Angular, RESTful Web Services and Spring Data JPA
  + Implement an event triggered in angular that executes a restful web service, which in turn invoke Spring Data JPA Repository that gets or persists data in database

**Integrate Spring Security with database**  
  
In "Spring RESTful Web Services" module, we implemented Spring Security with in memory database. In this hands on we will integrate Spring Security to use Spring Data JPA repository that manages the user details.  
  
**Prepare the Schema and Repository classes**

* Create relevant tables for storing user and role in ormlearn schema. (Refer the ER diagram in the end)
* Create classes User and Role with necessary many-to-many mapping
* Define roleList in User with eager fetch
* Create UserRepository with inclusion of a Query Method 'findByUsername' to get an user based on username.
* In user table create two users one for user and one for admin
* Make the password as $2a$10$R/lZJuT9skteNmAku9Y7aeutxbOKstD5xE5bHOf74M2PHZipyt3yK, which is the bcrypt encoding of the text 'pwd'
* Test if findByUsername() method works fine and retrieves the user and roles

​​​​​​​

**AppUser.java**  
This new bean class has to be implement UserDetails class of spring security framework, so that spring framework can read user details from this bean. Refer steps below to implement.

* Create bean class AppUser to hold the user details. This needs to be implement UserDetails class of spring security framework, so that spring framework can read user details from this bean. Refer steps below to implement:
* Right click on com.cognizant.spring-learn.security > New > Class
* Enter "Name" as "AppUser"
* In "Interfaces" click "Add.." and select "org.springframework.security.core.userdetails.UserDetails"
* Click Finish
* The above steps will create AppUser class and populate the code with all the methods that needs to be implemented.
* Include below instance variables in this class:

    private User user; // entity reference

    private Collection<? extends GrantedAuthority> authorities; // to store role details

* Create a constructor with user as parameter. Obtain the roles from the user and convert them to authorities. Refer code below for authorities conversion that needs to be included in the constructor.

        this.authorities = user.getRoleList().stream()

                .map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());

* The above code converts the list of roles into a list of SimpleGrantedAuthority.
* Return authorities in getAuthorities() method
* Return user.getPassword() in getPassword() method
* Return true in all the below methods:
  + isAccountNonExpired()
  + isAccountNonLocked()
  + isCredentialsNonExpired()
  + isEnabled()

**AppUserDetailsService.java**

* Create a new class AppUserDetailsService in com.cognizant.spring-learn.security package.
* This class has to implement org.springframework.security.core.userdetails.UserDetailsService, so that spring security framework uses this class to retrieve user details.
* Autowire UserRepository
* Implement loadUserByUsername() method

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException

* Import UsernameNotFoundException from org.springframework.security.core.userdetails;
* Invoke findByUsername() method from UserRepository and get the user.
* If user is null throw UsernameNotFoundException
* If user is not null then create an instance of AppUser passing user obtained from repository.
* Return the AppUser reference.
* Test the execution of loadUserByUsername() to see if user details are retrieved along with roles.

**SecurityConfig.java**

* Autowire AppUserDetailsService
* In configure() method, comment out the in memory definition and bind spring framework to use the appUserDetailsService for authentication.

auth.userDetailsService(appUserDetailsService).passwordEncoder(passwordEncoder());

* Test the authentication web service using curl or postman.
* **A Unit Test should be self sufficient**  
    
  As the name "Unit Testing" suggests, it is supposed to test a specific method in isolation without worrying about the dependencies.  
    
  Let us try to understand this with an example. Consider a Service method that calls a method in Repository. The service method is dependent on repository and the repository is dependent on database. When we have to test a service method, we should test it without the dependency on repository.  
    
  A good unit test should be isolated. Avoid dependencies such as environment settings, register values, or databases.  
    
  Let us consider the loadUserByUsername() method we implemented recently.  
    
  In this code below, find out the lines that has dependency on repository and database.
* public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
* LOGGER.info("Start");
* LOGGER.debug("UserRepository:{}", userRepository);
* User user = userRepository.findByUsername(username);
* LOGGER.debug("User:{}", user);
* if (user == null) {
* throw new UsernameNotFoundException(username);
* }
* LOGGER.info("End");
* return new AppPrincipalUser(user);
* }
* SME to walkthrough each line of the above code and identify the lines that has dependency.  
    
  In the subsequent hands on we will identify how to avoid the dependency using mockito.

**Implement integrated testing of loadUserByUsername() method**

* Open spring-learn application in Eclipse
* Create a new package com.cognizant.spring-learn.service in src/test/java
* Create a new class UserDetailsServiceTest
* Define the following annotations at class level. This ensures that the class has spring application context.

@RunWith(SpringRunner.class)

@SpringBootTest

* Autowire UserDetailsService and implement the below test method. This method validates if the password returned is as per what is available in the database.

    @Test

    public void testLoadByUserName() {

        LOGGER.info("Start");

        UserDetails user = userDetailsService.loadUserByUsername("usr");

        String expected = "$2a$10$R/lZJuT9skteNmAku9Y7aeutxbOKstD5xE5bHOf74M2PHZipyt3yK";

        assertEquals(expected, user.getPassword());

        LOGGER.info("End");

    }

* Execute this as JUnit Test Case by right clicking on this test class and selecting Run > JUnit Test
* Check the logs to see that the service calls repository and it retrieves the data from database
* Ask yourself the following questions:
  + Is this method executed in isolation?
  + Is this test unit testing?

This kind of testing is integration testing and not unit testing.  
  
In the next hands on we will see how to isolate the testing using Mockito.

**Implement isolated testing with Mockito**  
  
Refer the code of the method to be tested:

    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        LOGGER.info("Start");

        LOGGER.debug("UserRepository:{}", userRepository);

        User user = userRepository.findByUsername(username);

        LOGGER.debug("User:{}", user);

        if (user == null) {

            throw new UsernameNotFoundException(username);

        }

        LOGGER.info("End");

        return new AppPrincipalUser(user);

    }

Our unit test cases should ensure that all the lines and all possible scenarios are tested.  
  
The above method can be tested in two scenarios:

* The method findByUsername() returns a valid
* The method findByUsername() returns null due to non-availability of user in the database

**Implement the steps below to test the first scenario:**

* Create a new test class UserDetailsServiceMockTest in src/test/java, com.cognizant.spring-learn.service package
* Annotate @RunWith with MockitoJUnitRunner.class (import org.mockito.junit.MockitoJUnitRunner)
* Annotate @SpringBootTest
* Include test method mockTestLoadUserByUsername()
* Implement following steps in the test method.
* Using Mockit.mock() method create a mock of the repository

UserRepository repository = Mockito.mock(UserRepository.class);

* Using below line of code define what happens when findByUsername() method in repository is called. This is how we are isolating the dependency.

when(repository.findByUsername("usr")).thenReturn(createUser());

* Due to the above defintion the database call will not happen, we need to simulate the return of User as part of this call. We achieve this by using a new createUser() method that takes care of creating a user instance with necessary password and role that is expected. Implement this method within the test class.
* Create a new constructor in AppUserDetailsService that accepts UserRepository as input and sets the userRepository instance variable.
* Create AppUserDetailsService instance using the mocked repository

AppUserDetailsService service = new AppUserDetailsService(repository); // repository refers to the mock repository created

* Invoke the method we want to test. When this method is executed, the repository's find method will not be invoke, but mockito will just return whatever is returned by createUser() method.

UserDetails user = service.loadUserByUsername("usr");

* Include the lines below to assert if the password matches.

String expected = "$2a$10$R/lZJuT9skteNmAku9Y7aeutxbOKstD5xE5bHOf74M2PHZipyt3yK";

assertEquals(expected, user.getPassword());

* Run the class as JUnit test and check if the test passes.

**Implement the mockito test for the second scenario when user is returned as null**

* Mock the findByUsername() to return null
* Pass the test case if UsernameNotFoundException is thrown. This can be implemented by asserting true in the catch block of UsernameNotFoundException handling. Include return after assert.
* Assert it as false after catch block

**Code Coverage of JUnit Testing**  
  
Following steps below to check the test coverage of JUnit testing:

* Right click on test class
* Select "Coverage As" > "JUnit Test"
* This executes the following actions:
  + Executes the unit test cases
  + Marks each line as red and green to denote if a line was executed or not. Open various files of your choice and check the relevance of red and green lines.
  + Generates a coverage report that can be seen in a new view "Coverage". Maximize the "Coverage" view, expand the packages to view the class wise code coverage percentage.

**Integrate employee web services with Spring Data JPA**  
  
Hope you all remember the following features implemented as part of angular hands on:

* Display list of employees
* Clicking edit link on an employee will open employee form with populated values
* Clicking save on edit form to persist the employee details
* Clicking delete link on an employee has to delete the employee

Currently the above implementations are done with in memory static data.  
  
Using repository and appropriate beans implement this with repository and service.  
  
Test in the angular application to see if the features work end to end.

**Integrate employee web services with Spring Data JPA**  
  
Hope you all remember the following features implemented as part of angular hands on:

* Display list of employees
* Clicking edit link on an employee will open employee form with populated values
* Clicking save on edit form to persist the employee details
* Clicking delete link on an employee has to delete the employee

Currently the above implementations are done with in memory static data.  
  
Using repository and appropriate beans implement this with repository and service.  
  
Test in the angular application to see if the features work end to end.

**Implement end to end country search feature**  
  
Create a new component and route in angular application for searching country.  
  
This component should have a text box and a div in the bottom of the text box. This div will list the countries that matches with the keyword entered in text box. The countries within the div needs to be displayed line by line.  
  
Create necessary service in angular application that invokes the respective REST API created for returning the country list based on keyword.  
  
This REST API needs to be called on the key press event of the search text box.