**MANDATORY HANDS-ON**

**(ALSO INCLUDE ALL OTHER HANDS-ON OF WEEK-3)**

**WEEK-3**

**SPRING CORE AND MAVEN**

**Exercise 1: Configuring a Basic Spring Application**

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" ...>

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

</project>

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("BookRepository: Saving book - " + bookName);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println("BookService: Adding book - " + bookName);

bookRepository.saveBook(bookName);

}

}

**LibraryManagementApplication.java (Main.java)**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Spring in Action");

}

}

**Exercise 2: Implementing Dependency Injection**

**BookService.java**

public class BookService {

private BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println("BookService: Adding book - " + bookName);

bookRepository.saveBook(bookName);

}

}

**applicationContext.xml**

<bean id="bookService" class="com.library.service.BookService">

<constructor-arg ref="bookRepository"/>

</bean>

**Exercise 3: Implementing Logging with Spring AOP**

**pom.xml**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.30</version>

</dependency>

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.\*;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

@Around("execution(\* com.library.service.\*.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object value = joinPoint.proceed();

long end = System.currentTimeMillis();

System.out.println("[LOG] " + joinPoint.getSignature() + " executed in " + (end - start) + " ms");

return value;

}

}

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<context:component-scan base-package="com.library"/>

<aop:aspectj-autoproxy/>

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**Run LibraryManagementApplication.java**

**Expected Console Output:**

**[**LOG] void com.library.service.BookService.addBook(String) executed in 2 ms

BookService: Adding book - Spring in Action

BookRepository: Saving book - Spring in Action

**Exercise 4: Creating and Configuring a Maven Project**

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.30</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Exercise 5: Configuring the Spring IoC Container**

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("Saving book to DB: " + title);

}

}

**Main.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Effective Java");

}

}

**Exercise 6: Configuring Beans with Annotations**

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com.library" />

</beans>

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void saveBook(String title) {

System.out.println("Saving book to DB: " + title);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public void addBook(String title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

**LibraryManagementApplication.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean(BookService.class);

bookService.addBook("Clean Code");

}

}

**Exercise 7: Implementing Constructor and Setter Injection**

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Constructor Injection: BookRepository injected");

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Setter Injection: BookRepository injected");

}

public void addBook(String title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService">

<constructor-arg ref="bookRepository"/>

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

Run the app

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean(BookService.class);

service.addBook("Refactoring");

**Exercise 8: Implementing Basic AOP with Spring**

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.annotation.\*;

import org.aspectj.lang.JoinPoint;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

@Before("execution(\* com.library.service.\*.\*(..))")

public void beforeMethod(JoinPoint joinPoint) {

System.out.println("[Before] " + joinPoint.getSignature().getName() + " called.");

}

@After("execution(\* com.library.service.\*.\*(..))")

public void afterMethod(JoinPoint joinPoint) {

System.out.println("[After] " + joinPoint.getSignature().getName() + " completed.");

}

}

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<context:component-scan base-package="com.library" />

<aop:aspectj-autoproxy/>

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**Exercise 9: Creating a Spring Boot Application**

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

**application.properties**

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

**Book.java**

package com.library.entity;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

}

**BookRepository.java**

package com.library.repository;

import com.library.entity.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {}

**BookController.java**

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@PostMapping

public Book create(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping

public List<Book> getAll() {

return bookRepository.findAll();

}

}

**LibraryManagementApplication.java**

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

**SPRING DATA JPA WITH SPRING BOOT, HIBERNATE**

**Exercise 1: Employee Management System - Overview and Setup**

**Project Name: EmployeeManagementSystem**

**application.properties**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

**Exercise 2: Employee Management System - Creating Entities**

**Department Entity**

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department", cascade = CascadeType.ALL)

private List<Employee> employees = new ArrayList<>();

}

**Employee Entity**

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**Exercise 3: Employee Management System - Creating Repositories**

**EmployeeRepository**

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByName(String name);

}

**DepartmentRepository**

public interface DepartmentRepository extends JpaRepository<Department, Long> {

Optional<Department> findByName(String name);

}

**Exercise 4: Employee Management System - Implementing CRUD Operations**

**EmployeeController**

@RestController

@RequestMapping("/employees")

@RequiredArgsConstructor

public class EmployeeController {

private final EmployeeRepository employeeRepo;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepo.findAll();

}

@PostMapping

public Employee addEmployee(@RequestBody Employee employee) {

return employeeRepo.save(employee);

}

@PutMapping("/{id}")

public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmp) {

return employeeRepo.findById(id)

.map(emp -> {

emp.setName(updatedEmp.getName());

emp.setEmail(updatedEmp.getEmail());

emp.setDepartment(updatedEmp.getDepartment());

return ResponseEntity.ok(employeeRepo.save(emp));

}).orElse(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {

if (employeeRepo.existsById(id)) {

employeeRepo.deleteById(id);

return ResponseEntity.ok().build();

}

return ResponseEntity.notFound().build();

}

}

**DepartmentController**

@RestController

@RequestMapping("/departments")

@RequiredArgsConstructor

public class DepartmentController {

private final DepartmentRepository departmentRepo;

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepo.findAll();

}

@PostMapping

public Department addDepartment(@RequestBody Department department) {

return departmentRepo.save(department);

}

@PutMapping("/{id}")

public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department updatedDept) {

return departmentRepo.findById(id)

.map(dept -> {

dept.setName(updatedDept.getName());

return ResponseEntity.ok(departmentRepo.save(dept));

}).orElse(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {

if (departmentRepo.existsById(id)) {

departmentRepo.deleteById(id);

return ResponseEntity.ok().build();

}

return ResponseEntity.notFound().build();

}

}

**Exercise 5: Employee Management System - Defining Query Methods**

**In EmployeeRepository**

List<Employee> findByEmailContaining(String keyword);

@Query("SELECT e FROM Employee e WHERE e.name LIKE %:name%")

List<Employee> searchByName(@Param("name") String name);

**In DepartmentRepository**

List<Department> findByNameContaining(String name);

**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

**Pagination with Pageable**

@GetMapping("/paged")

public Page<Employee> getEmployeesPaged(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "5") int size,

@RequestParam(defaultValue = "id") String sortBy

) {

Pageable pageable = PageRequest.of(page, size, Sort.by(sortBy));

return employeeRepo.findAll(pageable);

}

**Exercise 7: Employee Management System - Enabling Entity Auditing**

**Enable Auditing**

**JpaConfig.java**

@Configuration

@EnableJpaAuditing

public class JpaConfig {}

**BaseEntity.java**

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

@Getter

@Setter

public abstract class BaseEntity {

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime modifiedDate;

}

**Update Employee and Department Entities**

@Entity

public class Employee extends BaseEntity {

}

@Entity

public class Department extends BaseEntity {

}

**Enable auditing in application.properties**

spring.jpa.properties.javax.persistence.validation.mode=none

**Exercise 8: Employee Management System - Creating Projections**

**Interface-Based Projection**

**EmployeeSummary.java**

public interface EmployeeSummary {

String getName();

String getEmail();

}

**Add to EmployeeRepository.java**

List<EmployeeSummary> findByDepartmentId(Long departmentId);

**Class-Based Projection**

**EmployeeDTO.java**

public class EmployeeDTO {

private String name;

private String email;

public EmployeeDTO(String name, String email) {

this.name = name;

this.email = email;

}

public String getName() {

return name;

}

public String getEmail() {

return email;

}

}

**Add to EmployeeRepository.java**

@Query("SELECT new com.example.EmployeeDTO(e.name, e.email) FROM Employee e WHERE e.department.id = :deptId")

List<EmployeeDTO> fetchByDepartment(@Param("deptId") Long deptId);

**Exercise 9: Employee Management System - Customizing Data Source Configuration**

**application.properties:**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

For **externalized config**, make it dynamic:

spring.datasource.url=${DB\_URL:jdbc:h2:mem:testdb}

spring.datasource.username=${DB\_USERNAME:sa}

spring.datasource.password=${DB\_PASSWORD:password}

**Exercise 10: Employee Management System - Hibernate-Specific Features**

**application.properties**

spring.jpa.properties.hibernate.jdbc.batch\_size=30

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true

**Employee.java**

@Entity

@BatchSize(size = 30)

@DynamicUpdate

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**EmployeeService.java**

@Service

@RequiredArgsConstructor

public class EmployeeService {

private final EmployeeRepository employeeRepo;

@Transactional

public void batchInsert(List<Employee> employees) {

for (int i = 0; i < employees.size(); i++) {

employeeRepo.save(employees.get(i));

if (i % 30 == 0) {

employeeRepo.flush();

}

}

}

}

**Hands on 1**

**Spring Data JPA - Quick Example**

MySQL Setup

CREATE SCHEMA ormlearn;

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India'), ('US', 'United States of America');

**application.properties**

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

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

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

spring.jpa.hibernate.ddl-auto=validate

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

**Country.java**

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

public String getCode() { return code; }

public void setCode(String code) { this.code = code; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**CountryRepository.java**

package com.cognizant.ormlearn.repository;

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> {}

**CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java**

package com.cognizant.ormlearn;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@SpringBootApplication

public class OrmLearnApplication {

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

private static CountryService countryService;

public static void main(String[] args) {

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

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

}

**Hands on 2**

**Hibernate XML Config implementation walk through**

**Employee.hbm.xml**

<hibernate-mapping>

<class name="Employee" table="EMPLOYEE">

<id name="id" column="ID">

<generator class="native"/>

</id>

<property name="firstName" column="FIRST\_NAME"/>

<property name="lastName" column="LAST\_NAME"/>

<property name="salary" column="SALARY"/>

</class>

</hibernate-mapping>

**Hands on 3**

**Hibernate Annotation Config implementation walk through**   
@Entity

@Table(name = "EMPLOYEE")

public class Employee {

@Id

@GeneratedValue

@Column(name = "ID")

private int id;

@Column(name = "FIRST\_NAME")

private String firstName;

@Column(name = "LAST\_NAME")

private String lastName;

@Column(name = "SALARY")

private int salary;

}

**hibernate.cfg.xml**

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/yourdb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

...

</session-factory>

</hibernate-configuration>

**Hands on 4**

**Difference between JPA, Hibernate and Spring Data JPA**

Hibernate

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

session.save(employee);

tx.commit();

session.close();

Spring Data JPA

@Autowired

private EmployeeRepository repo;

@Transactional

public void addEmployee(Employee employee) {

repo.save(employee);

}

**Hands on 5**

**Implement services for managing Country**   
CountryRepository.java

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String partialName);

}

**Hands on 6**

**Find a country based on country code**   
CountryService.java

@Transactional

public Country findCountryByCode(String code) throws CountryNotFoundException {

return countryRepository.findById(code)

.orElseThrow(() -> new CountryNotFoundException("Country not found"));

}

CountryNotFoundException.java

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

OrmLearnApplication.java

private static void testFindCountryByCode() throws Exception {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

**Hands on 7**

**Add a new country**

CountryService.java

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

OrmLearnApplication.java

private static void testAddCountry() throws Exception {

Country country = new Country();

country.setCode("XY");

country.setName("Testland");

countryService.addCountry(country);

Country saved = countryService.findCountryByCode("XY");

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

}

**Hands on 8**

**Update a country based on code**

CountryService.java

@Transactional

public void updateCountry(String code, String name) throws CountryNotFoundException {

Country country = countryRepository.findById(code)

.orElseThrow(() -> new CountryNotFoundException("Country not found"));

country.setName(name);

countryRepository.save(country);

}

OrmLearnApplication.java

private static void testUpdateCountry() throws Exception {

countryService.updateCountry("XY", "New Testland");

}

**Hands on 9**

**Delete a country based on code**

CountryService.java

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

OrmLearnApplication.java

private static void testDeleteCountry() {

countryService.deleteCountry("XY");

}

**Hands on 1**

**Write queries on country table using Query Methods**

Matching country names by substring

List<Country> findByNameContainingIgnoreCase(String keyword);

Matching with sorting

List<Country> findByNameContainingIgnoreCaseOrderByNameAsc(String keyword);

**Find all countries starting with a specific letter**

List<Country> findByNameStartingWithIgnoreCase(String prefix);

**Hands on 2**

**Write queries on stock table using Query Methods**

Repository Interface

public interface StockRepository extends JpaRepository<Stock, Integer> {

List<Stock> findByCodeAndDateBetween(String code, LocalDate startDate, LocalDate endDate);

List<Stock> findByCodeAndCloseGreaterThan(String code, BigDecimal close);

List<Stock> findTop3ByOrderByVolumeDesc();

List<Stock> findTop3ByCodeOrderByCloseAsc(String code);

}

OrmLearnApplication

@Autowired

private StockRepository stockRepository;

private static void testFacebookStocksSeptember2019() {

LocalDate start = LocalDate.of(2019, 9, 1);

LocalDate end = LocalDate.of(2019, 9, 30);

List<Stock> stocks = stockRepository.findByCodeAndDateBetween("FB", start, end);

stocks.forEach(System.out::println);

}

**Hands on 3**

**Create payroll tables and bean mapping**

Employee.java

@Entity

public class Employee {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

private double salary;

private boolean permanent;

@Column(name = "em\_date\_of\_birth")

private Date dateOfBirth;

@ManyToOne

@JoinColumn(name = "em\_dp\_id")

private Department department;

@ManyToMany(fetch = FetchType.EAGER)

@JoinTable(name = "employee\_skill",

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

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

private Set<Skill> skillList;

}

Department.java

@Entity

public class Department {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

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

private Set<Employee> employeeList;

}

Skill.java

@Entity

public class Skill {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

@ManyToMany(mappedBy = "skillList")

private Set<Employee> employeeList;

}

**Hands on 4**

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

OrmLearnApplication.java

private static void testGetEmployee() {

Employee emp = employeeService.get(1);

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

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

}

private static void testAddEmployee() {

Employee emp = new Employee();

emp.setName("John");

emp.setSalary(50000);

emp.setPermanent(true);

emp.setDateOfBirth(new Date());

Department dept = departmentService.get(1);

emp.setDepartment(dept);

employeeService.save(emp);

}

**Hands on 5**

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

OrmLearnApplication.java

private static void testGetDepartment() {

Department dept = departmentService.get(1);

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

LOGGER.debug("Employees:{}", dept.getEmployeeList());

}

**Hands on 6**

**Implement many to many relationship between Employee and Skill**

OrmLearnApplication.java

private static void testAddSkillToEmployee() {

Employee emp = employeeService.get(1);

Skill skill = skillService.get(3);

Set<Skill> skillList = emp.getSkillList();

skillList.add(skill);

employeeService.save(emp);

}

**Hands on 1**

**Introduction to HQL and JPQL** 

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

**Hands on 2**

**Get all permanent employees using HQL**   
@Query("SELECT e FROM Employee e LEFT JOIN FETCH e.department d LEFT JOIN FETCH e.skillList WHERE e.permanent = 1")

List<Employee> getAllPermanentEmployees();

**Hands on 3**

**Fetch quiz attempt details using HQL**

@Query("SELECT a FROM Attempt a JOIN FETCH a.user u JOIN FETCH a.attemptQuestions aq JOIN FETCH aq.question q JOIN FETCH aq.attemptOptions ao JOIN FETCH ao.option o WHERE u.id = :userId AND a.id = :attemptId")

Attempt getAttempt(@Param("userId") int userId, @Param("attemptId") int attemptId);

**Hands on 4**

**Get average salary using HQL**

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

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

**Hands on 5**

**Get all employees using Native Query**   
@Query(value = "SELECT \* FROM employee", nativeQuery = true)

List<Employee> getAllEmployeesNative();

**Hands on 6**

**Criteria Query**

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<Product> cq = cb.createQuery(Product.class);

Root<Product> product = cq.from(Product.class);

List<Predicate> predicates = new ArrayList<>();

if (cpu != null) predicates.add(cb.equal(product.get("cpu"), cpu));

if (ram != null) predicates.add(cb.equal(product.get("ram"), ram));

cq.where(predicates.toArray(new Predicate[0]));

**Exercise 1: Employee Management System - Overview and Setup**

**application.properties**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.show-sql=true

spring.h2.console.enabled=true

Main Application Class

@SpringBootApplication

@EnableJpaAuditing

public class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

**Exercise 2: Employee Management System - Creating Entities**

Department.java

@Entity

@EntityListeners(AuditingEntityListener.class)

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department", cascade = CascadeType.ALL)

private List<Employee> employees;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime modifiedDate;

}

Employee.java

@Entity

@EntityListeners(AuditingEntityListener.class)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime modifiedDate;

}

**Exercise 3: Employee Management System - Creating Repositories**

DepartmentRepository.java

public interface DepartmentRepository extends JpaRepository<Department, Long> {

List<Department> findByNameContaining(String name);

}

EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByNameContaining(String name);

@Query("SELECT e FROM Employee e WHERE e.email = :email")

Employee findByEmail(@Param("email") String email);

@Query("SELECT e.name AS name, e.email AS email FROM Employee e")

List<EmployeeNameEmailProjection> fetchNameEmailOnly();

Page<Employee> findByDepartmentId(Long departmentId, Pageable pageable);

}

**Exercise 4: Employee Management System - Implementing CRUD Operations**

DepartmentController.java

@RestController

@RequestMapping("/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository repo;

@GetMapping

public List<Department> getAll() {

return repo.findAll();

}

@PostMapping

public Department save(@RequestBody Department d) {

return repo.save(d);

}

@DeleteMapping("/{id}")

public void delete(@PathVariable Long id) {

repo.deleteById(id);

}

}

EmployeeController.java

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository repo;

@GetMapping

public List<Employee> getAll() {

return repo.findAll();

}

@GetMapping("/{id}")

public Employee getById(@PathVariable Long id) {

return repo.findById(id).orElseThrow();

}

@PostMapping

public Employee save(@RequestBody Employee e) {

return repo.save(e);

}

@PutMapping("/{id}")

public Employee update(@PathVariable Long id, @RequestBody Employee e) {

e.setId(id);

return repo.save(e);

}

@DeleteMapping("/{id}")

public void delete(@PathVariable Long id) {

repo.deleteById(id);

}

}

**Exercise 5: Employee Management System - Defining Query Methods**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByNameContaining(String name);

@Query("SELECT e FROM Employee e WHERE e.email = :email")

Employee findByEmail(@Param("email") String email);

@Query("SELECT e.name AS name, e.email AS email FROM Employee e")

List<EmployeeNameEmailProjection> fetchNameEmailOnly();

Page<Employee> findByDepartmentId(Long departmentId, Pageable pageable);

}

DepartmentRepository.java

public interface DepartmentRepository extends JpaRepository<Department, Long> {

List<Department> findByNameContaining(String name);

}

**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

**EmployeeController.java**

@GetMapping("/search")

public Page<Employee> search(@RequestParam Long deptId,

@RequestParam int page,

@RequestParam int size,

@RequestParam String sort) {

return repo.findByDepartmentId(deptId, PageRequest.of(page, size, Sort.by(sort)));

}**Top of Form**

**Exercise 7: Employee Management System - Enabling Entity Auditing**

**EmployeeManagementSystemApplication.java**

@SpringBootApplication

@EnableJpaAuditing

public class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

Common Annotations in Employee and Department entities

@EntityListeners(AuditingEntityListener.class)

@CreatedDate

@LastModifiedDate

Employee.java

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime modifiedDate;

**Bottom of Form**

**Exercise 8: Employee Management System - Creating Projections**

EmployeeNameEmailProjection.java

public interface EmployeeNameEmailProjection {

String getName();

String getEmail();

}

EmployeeController.java

@GetMapping("/projections")

public List<EmployeeNameEmailProjection> getProjections() {

return repo.fetchNameEmailOnly();

}

**Exercise 9: Employee Management System - Customizing Data Source Configuration**

application.properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

**Exercise 10: Employee Management System - Hibernate-Specific Features**

**application.properties**

spring.jpa.properties.hibernate.jdbc.batch\_size=30

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true