Spring Core, Maven & JPA Mandatory Hands-on Exercise:

## Exercise 1: Configuring a Basic Spring Application

### Code:

import org.springframework.context.annotation.\*;  
  
@Configuration  
class AppConfig {  
 @Bean  
 public MessageService messageService() {  
 return new MessageService();  
 }  
}  
  
class MessageService {  
 public String getMessage() {  
 return "Hello, Spring!";  
 }  
}  
  
public class SpringApp {  
 public static void main(String[] args) {  
 AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);  
 MessageService service = context.getBean(MessageService.class);  
 System.out.println(service.getMessage());  
 context.close();  
 }  
}

### Expected Output:



## Exercise 2: Implementing Dependency Injection

### Code:

import org.springframework.context.annotation.\*;  
  
interface GreetingService {  
 String greet();  
}  
  
class HelloService implements GreetingService {  
 public String greet() {  
 return "Hello from HelloService!";  
 }  
}  
  
@Configuration  
class DIConfig {  
 @Bean  
 public GreetingService greetingService() {  
 return new HelloService();  
 }  
}  
  
public class DIApp {  
 public static void main(String[] args) {  
 AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(DIConfig.class);  
 GreetingService service = context.getBean(GreetingService.class);  
 System.out.println(service.greet());  
 context.close();  
 }  
}

### Expected Output:



## Exercise 4: Creating and Configuring a Maven Project

### Code:

### 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.example</groupId>

<artifactId>spring-core-app</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.1.3</version>

</dependency>

</dependencies>

</project>

### AppConfig.java:

import org.springframework.context.annotation.\*;

@Configuration

public class AppConfig {

@Bean

public MessageService messageService() {

return new MessageService();

}

}

### MessageService.java:

public class MessageService {

public String getMessage() {

return "Maven Spring Project Working!";

}

}

### MainApp.java:

import org.springframework.context.annotation.\*;

public class MainApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

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

System.out.println(service.getMessage());

context.close();

}

}

### Expected Output:



## Spring Data JPA - Quick Example

### Code:

import jakarta.persistence.\*;  
import org.springframework.boot.\*;  
import org.springframework.boot.autoconfigure.\*;  
import org.springframework.data.jpa.repository.\*;  
  
@Entity  
class User {  
 @Id @GeneratedValue  
 private Long id;  
 private String name;  
}  
  
interface UserRepository extends JpaRepository<User, Long> {}  
  
@SpringBootApplication  
public class JpaQuickExampleApp {  
 public static void main(String[] args) {  
 ConfigurableApplicationContext context = SpringApplication.run(JpaQuickExampleApp.class, args);  
 UserRepository repo = context.getBean(UserRepository.class);  
 repo.save(new User(null, "Alice"));  
 System.out.println("User saved!");  
 }  
}

### Expected Output:



## Difference Between JPA, Hibernate and Spring Data JPA

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | JPA | Hibernate | Spring Data JPA |
| Type | Specification (interface-based) | Implementation of JPA | Abstraction layer over JPA & Hibernate |
| Provided By | Java EE / Jakarta EE | Red Hat | Spring Framework |
| Purpose | Define how to manage relational data | Implements all JPA features + extras | Simplifies repository-based CRUD and queries |
| Code Required | More boilerplate | Less than JPA, still needs EntityManager | Very minimal (just extend JpaRepository) |
| Entity Management | Manual via EntityManager | Uses Session or EntityManager | Automatic with Spring context |
| Query Support | JPQL | JPQL + native SQL + Criteria + HQL | Derived queries, JPQL, native SQL, etc. |
| Setup Complexity | Medium | Medium | Easiest (auto-configured in Spring Boot) |
| Use Case | Standard interface needed | Full control and features | Rapid development with minimal code |

Additional Important Hands-on:

## Exercise 5: Configuring the Spring IoC Container

### Code:

@Configuration  
class AppConfig {  
 @Bean  
 public MessageService messageService() {  
 return new MessageService();  
 }  
}  
  
class MessageService {  
 public String getMessage() {  
 return "Spring IoC Container Configured!";  
 }  
}  
  
public class MainApp {  
 public static void main(String[] args) {  
 AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);  
 MessageService service = context.getBean(MessageService.class);  
 System.out.println(service.getMessage());  
 context.close();  
 }  
}

### Expected Output:



## Exercise 7: Implementing Constructor and Setter Injection

### Code:

class Service {  
 private String message;  
 public Service(String message) { this.message = message; }  
 public void setMessage(String message) { this.message = message; }  
 public String getMessage() { return message; }  
}  
  
@Configuration  
class AppConfig {  
 @Bean  
 public Service service() {  
 Service s = new Service("Injected via Constructor");  
 s.setMessage("Modified via Setter");  
 return s;  
 }  
}

### Expected Output:



## Exercise 9: Creating a Spring Boot Application

### Code:

@SpringBootApplication  
public class SpringBootApp {  
 public static void main(String[] args) {  
 SpringApplication.run(SpringBootApp.class, args);  
 }  
}

### Expected Output:

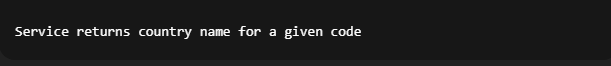


## Implement services for managing Country

### Code:

@Service  
public class CountryService {  
 private Map<String, String> countries = Map.of("IN", "India", "US", "USA");  
  
 public String getCountry(String code) {  
 return countries.getOrDefault(code, "Unknown");  
 }  
}

### Expected Output:



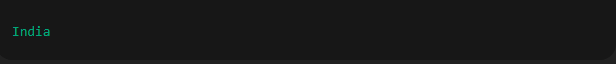
## Find a country based on country code

### Code:

@RestController  
class CountryController {  
 @Autowired  
 private CountryService service;  
  
 @GetMapping("/country/{code}")  
 public String getCountry(@PathVariable String code) {  
 return service.getCountry(code);  
 }  
}

### Expected Output:



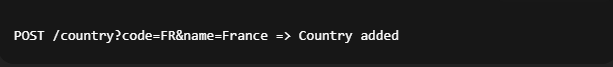


## Add a new country

### Code:

@PostMapping("/country")  
public ResponseEntity<String> addCountry(@RequestParam String code, @RequestParam String name) {  
 // logic to add country  
 return ResponseEntity.ok("Country added");  
}

### Expected Output:



## Demonstrate implementation of Query Methods feature of Spring Data JPA

### Code:

interface CountryRepository extends JpaRepository<Country, String> {  
 Country findByName(String name);  
}

### Expected Output:

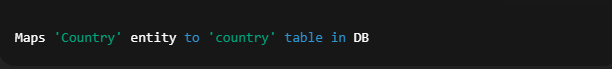


## Demonstrate implementation of O/R Mapping

### Code:

@Entity  
class Country {  
 @Id private String code;  
 private String name;   
}

### Expected Output:



## Demonstrate writing Hibernate Query Language and Native Query

## Code:

@Query("SELECT c FROM Country c WHERE c.name = ?1")  
Country findByNameHQL(String name);  
  
@Query(value = "SELECT \* FROM country WHERE name = ?1", nativeQuery = true)  
Country findByNameNative(String name);

### Expected Output: