Spring Core and Maven :-

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

applicationContext.xml :-  
<?xml version="1.0" encoding="UTF-8"?>  
<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"/>  
</beans>

BookRepository.java :-  
package com.library.repository;   
  
public class BookRepository {  
    public void saveBook() {  
        System.out.println("Book saved to repository.");  
    }  
}

BookService.java :-  
package com.library.service;   
  
public class BookService {  
    public void addBook() {  
        System.out.println("Book added via service.");  
    }  
}

LibraryManagementApplication.java :-  
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();  
    }  
}  
Output :  
Book added via service.

Exercise 2 : **Implementing Dependency Injection**

Update applicationContext.xml for DI :-  
<bean id="bookRepository" class="com.library.repository.BookRepository"/>  
<bean id="bookService" class="com.library.service.BookService">  
    <property name="bookRepository" ref="bookRepository"/>  
</bean>   
  
Update BookService.java :-  
package com.library.service;   
  
import com.library.repository.BookRepository;   
  
public class BookService {  
    private BookRepository bookRepository;   
  
    // Setter for DI  
    public void setBookRepository(BookRepository bookRepository) {  
        this.bookRepository = bookRepository;  
    }   
  
    public void addBook() {  
        System.out.println("Book added via service.");  
        bookRepository.saveBook();  
    }  
}  
Output :-  
Book added via service.  
Book saved to repository.

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

1. Adding SpringAOP and WebMVC dependencies :-  
<dependencies>  
  <!-- Spring Context -->  
  <dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-context</artifactId>  
    <version>5.3.32</version>  
  </dependency>   
  
  <!-- Spring AOP -->  
  <dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-aop</artifactId>  
    <version>5.3.32</version>  
  </dependency>   
  
  <!-- Spring WebMVC -->  
  <dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-webmvc</artifactId>  
    <version>5.3.32</version>  
  </dependency>  
</dependencies>   
  
2. Configuring Maven Compiler Plugin :-  
<build>  
  <plugins>  
    <plugin>  
      <artifactId>maven-compiler-plugin</artifactId>  
      <version>3.10.1</version>  
      <configuration>  
        <source>1.8</source>  
        <target>1.8</target>  
      </configuration>  
    </plugin>  
  </plugins>  
</build>   
  
Output :  
Book added via service.  
Book saved to repository.

Spring Data JPA, Hibernate :-

MySQL Schema :-  
CREATE SCHEMA ormlearn;   
  
CREATE TABLE country (  
  co\_code VARCHAR(2) PRIMARY KEY,  
  co\_name VARCHAR(50)  
);   
  
INSERT INTO country VALUES ('IN', 'India');  
INSERT INTO country VALUES ('US', 'United States of America');   
  
application.properties :-  
# Logging  
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   
  
# DB 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  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect   
  
Country Entity :-  
@Entity  
@Table(name = "country")  
public class Country {   
  
    @Id  
    @Column(name = "co\_code")  
    private String code;   
  
    @Column(name = "co\_name")  
    private String name;   
  
    // Getters, Setters, toString  
}   
  
Country Repository :-  
@Repository  
public interface CountryRepository extends JpaRepository<Country, String> {  
}   
  
Country Service :-  
@Service  
public class CountryService {   
  
    @Autowired  
    private CountryRepository countryRepository;   
  
    @Transactional  
    public List<Country> getAllCountries() {  
        return countryRepository.findAll();  
    }  
}   
  
ORMLearnApplication.java :-  
@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");  
    }  
}   
  
Maven Build Command (with proxy) :   
  
mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456   
  
Q). Difference between JPA, Hibernate and Spring Data JPA.   
  
1. JPA (Java Persistence API)   
  
Purpose: Defines a standard for object-relational mapping (ORM) and managing relational data in Java applications.   
JPA itself is not an implementation. It defines interfaces like EntityManager, Query, EntityTransaction, etc.  
  
For example,  
  
public interface EntityManager {  
    void persist(Object entity);  
    <T> T find(Class<T> entityClass, Object primaryKey);  
}  
  
2. Hibernate   
  
Purpose: A popular ORM framework that implements the JPA specification (and adds extra features). You can use Hibernate either with or without JPA. Hibernate provides both JPA implementation and its native API. Its extra features are Caching (first and second-level), Criteria API (older Hibernate-style), etc.

3. Spring Data JPA   
  
Purpose: Simplifies JPA-based data access using Spring abstractions, Repository pattern, and auto query generation. It's a layer on top of JPA (and usually Hibernate underneath) that reduces boilerplate code. For example,  
  
public interface UserRepository extends JpaRepository<User, Long> {  
    List<User> findByLastName(String lastName);  
}   
  
Spring Data JPA will generate the SQL query automatically based on method name.