endava

UNIVERSITATEA DIN BUCUREȘTI
VIRTUTE ET SAPIENTIA

WEB TECHNOLOGIES USING JAVA

COURSE 10 – SPRING DATA JPA.

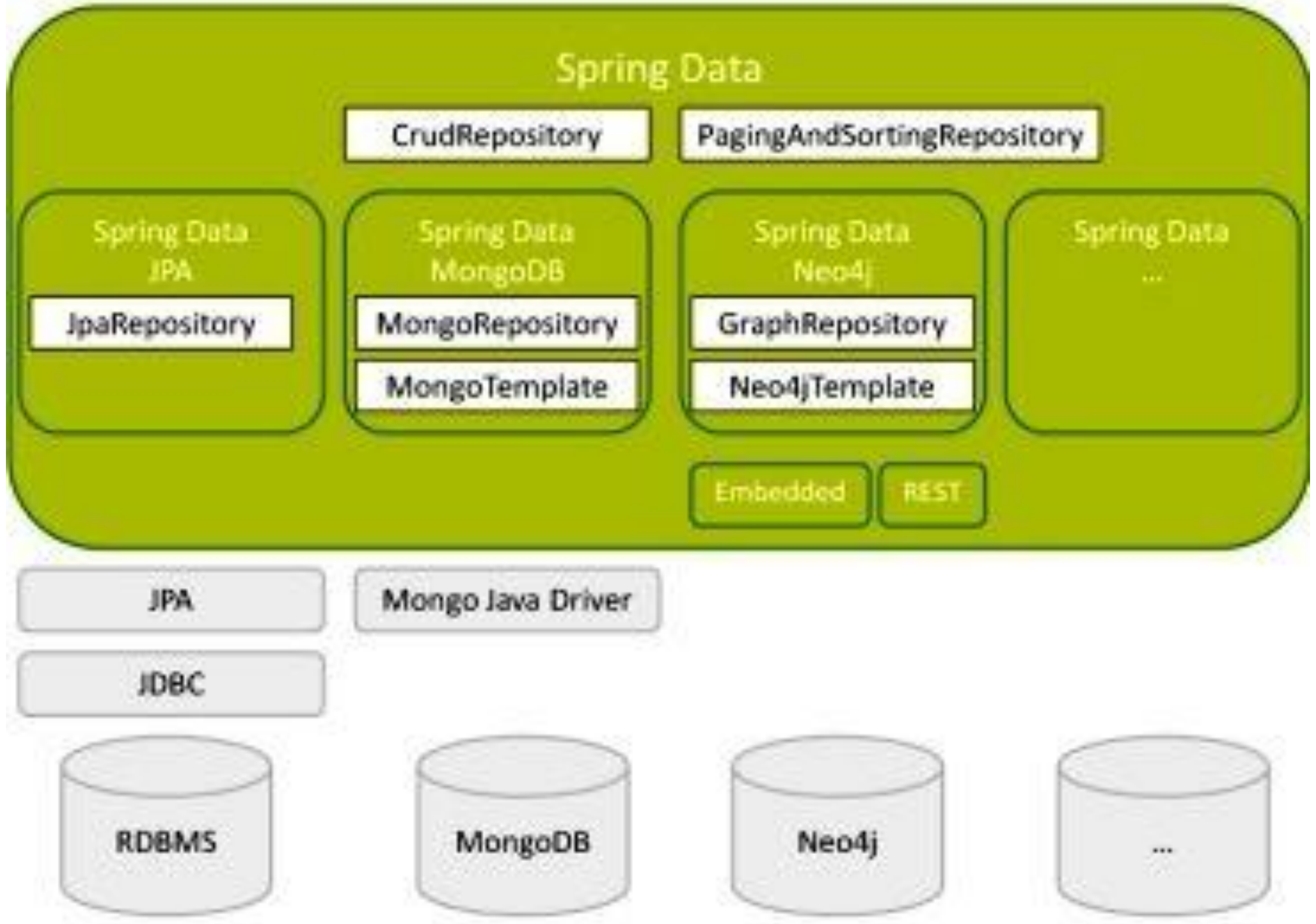
AGENDA

- SPRING DATA JPA
- TRANSACTION MANAGEMENT
- SPRING DATA JPA TRANSACTION MANAGEMENT

SPRING DATA JPA

- Spring Data significantly reduces the amount of boilerplate code required to implement data access layers for various persistence stores.
- Spring Data JPA is a JPA Data Access Abstraction.
- Spring Data JPA is not a JPA provider. It is a library/framework that adds an extra layer of abstraction on the top of a JPA provider (like Hibernate, Eclipse Link or any other JPA provider.).

SPRING DATA JPA

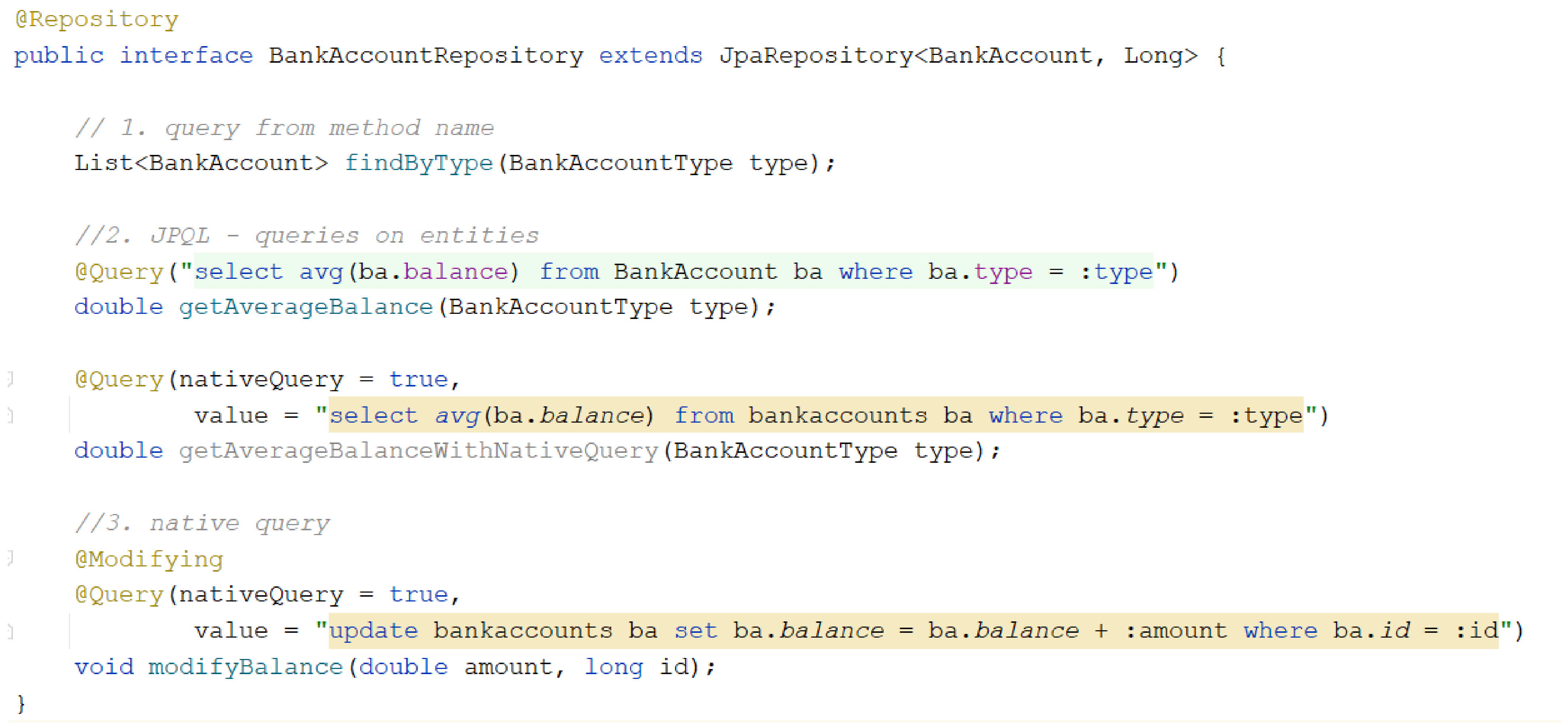


SPRING DATA JPA

- Queries built based on repository method names
- @Query with JPQL(Java Persistence Query Language)
- @Query(native = true)

- @Modifying

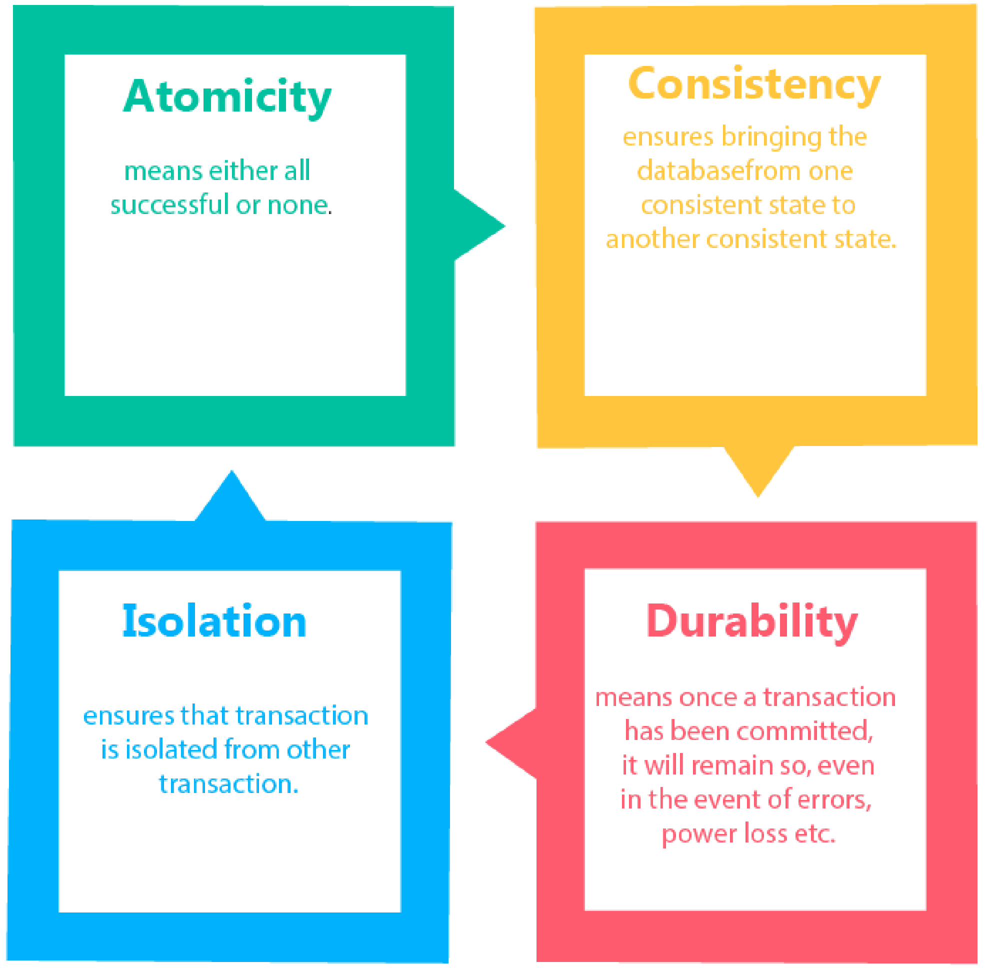
SPRING DATA JPA



```java
@Repository
public interface BankAccountRepository extends JpaRepository<BankAccount, Long> {

    // 1. query from method name
    List<BankAccount> findByType(BankAccountType type);

    //2. JPQL - queries on entities
    @Query("select avg(ba.balance)  from BankAccount ba where ba.type = :type")
    double getAverageBalance(BankAccountType type);

    @Query(nativeQuery = true,
            value = "select avg(ba.balance)  from bankaccounts ba where ba.type = :type")
    double getAverageBalanceWithNativeQuery(BankAccountType type);

    //3. native query
    @Modifying
    @Query(nativeQuery = true,
            value = "update bankaccounts ba set ba.balance = ba.balance + :amount where ba.id = :id")
    void modifyBalance(double amount, long id);
}
```

TRANSACTION MANAGEMENT

- transaction management aims to prevent data inconsistent state
- a transaction is a single logical unit of work that accesses and possibly modifies the contents of a database
- transaction properties are used for maintaining the integrity of database during transaction processing:
  - Atomicity
  - Consistency
  - Isolation
  - Durability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

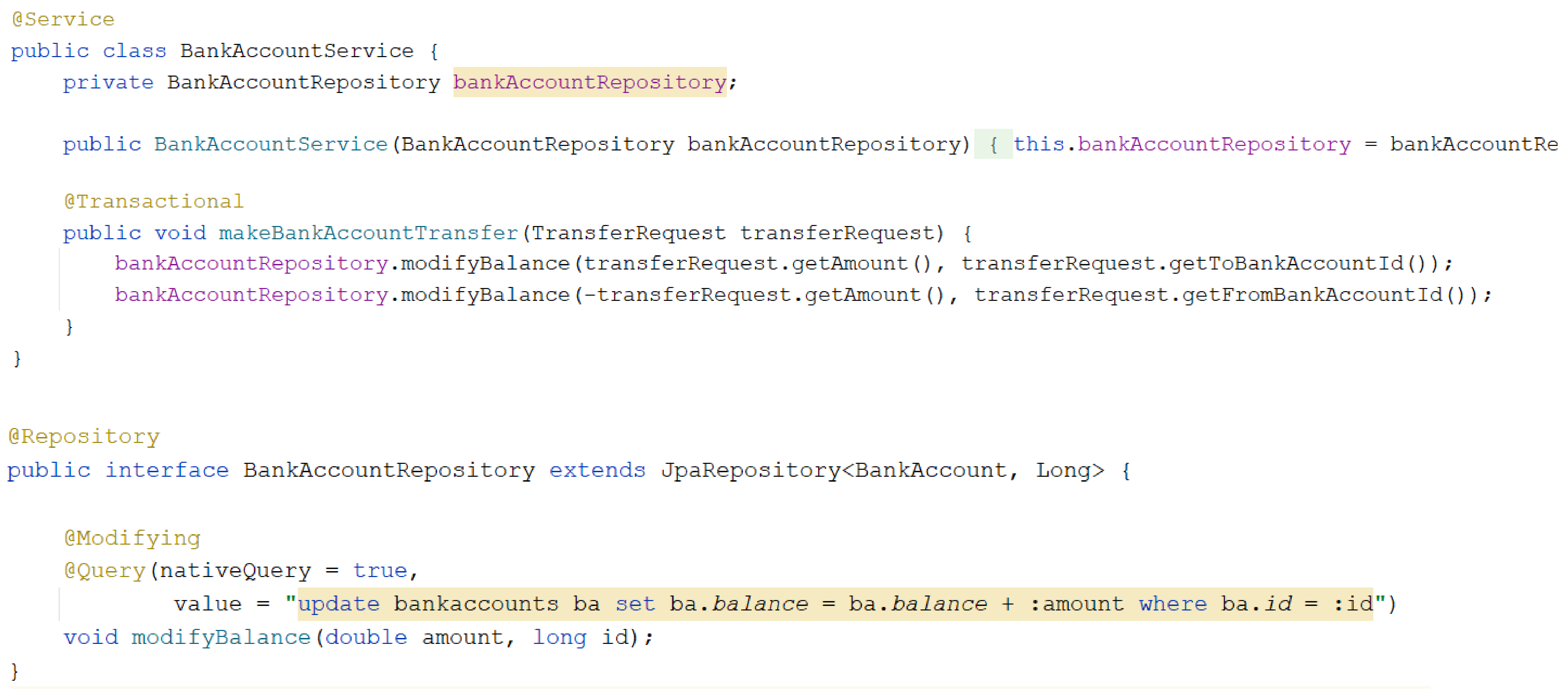
SPRING DATA JPA TRANSACTION MANAGEMENT

- @Transactional can be put on class or method level
- @Transactional attributes:
  - propagation: specifies how the transaction propagates. We can continue an existing transaction, if one exists, or create a new transaction. The default is Propagation.REQUIRED
  - readOnly: specifies if the transaction is read-only
  - noRollbackForClassName: specifies which exceptions should not cause a rollback of a transaction
  - rollbackForClassName: specifies which exceptions should cause a rollback of a transaction
  - timeout: specifies how long should we wait for a transaction to complete
  - isolation: specifies the isolation level for the transaction. The default is Isolation.DEFAULT
  - transactionManager: identifies the transaction manager that manages this transaction

SPRING DATA JPA TRANSACTION MANAGEMENT

- Spring Boot auto-configures a transaction manager based on the libraries in the classpath
- transaction managers for relational databases:
  - DataSourceTransactionManager: can manage transactions for one JDBC database resource
  - JpaTransactionManager: can manage transactions for one JPA database resource

SPRING DATA JPA



```
@Service
public class BankAccountService {
    private BankAccountRepository bankAccountRepository;

    public BankAccountService(BankAccountRepository bankAccountRepository) { this.bankAccountRepository = bankAccountRe

    @Transactional
    public void makeBankAccountTransfer(TransferRequest transferRequest) {
        bankAccountRepository.modifyBalance(transferRequest.getAmount(), transferRequest.getToBankAccountId());
        bankAccountRepository.modifyBalance(-transferRequest.getAmount(), transferRequest.getFromBankAccountId());
    }
}


@Repository
public interface BankAccountRepository extends JpaRepository<BankAccount, Long> {

    @Modifying
    @Query(nativeQuery = true,
            value = "update bankaccounts ba set ba.balance = ba.balance + :amount where ba.id = :id")
    void modifyBalance(double amount, long id);
}
```

BIBLIOGRAPHY

- Spring in Action, by Craig Walls
- Spring REST, by Balaji Varanasi, Sudha Belida
- Pro JPA 2, Mike Keith, Merrick Schincariol

- https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories
- https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repository-query-keywords
- https://docs.oracle.com/javaee/6/tutorial/doc/bnbtg.html

Q&A

THANK YOU

DANIELA SPILCĂ