1. **Онаследяване на repository**

* В parent interface се поставя анотация @NoRepositoryBean,

И екстендва JpaRepository

* В интерфейса, който екстендва parent-a се поставя анотация @Repository

@NoRepositoryBean

public interface IngredientRepository<T extends Ingredient> extends JpaRepository<T, Long>{

//…

}

@Repository

public interface ChemicalIngredientRepository extends IngredientRepository <BasicChemicalIngredient> {

List<ChemicalIngredient> findByChemicalFormula(String chemicalFormula);

}

BasicChemicalIngredient extends Ingredient

1. **@Modifying Query**

* Поставя се допълнителна анотация @Modifying, когато query-то извъшрва create, update ,delete. Асоциира се updateQuery, a без тази анотация е executeQuery.
* Поставя се **@Transactional** анотация на service-a, който извиква този метод за create, update, delete

1. **Custom Query - JOIN**

@Query(value = "select s from Shampoo s " +

"join s.ingredients i where i in :ingredients")

List<Shampoo> findByIngredientsIn(@Param(value = "ingredients")

Set<Ingredient> ingredients);

1. **Java-Based Configuration – в клас**

* **@Bean –** обект, който Spring изпозлва и ще даде възможност да се инджектва. Изпозлва се, когато не може да се променя дадения клас
* Поставя се анотация @Bean на желания getter за Spring class,

DataSource се използва за конфигуриране на spring database

* Може url, username, password, driverClass да се инджектват през конструктoр с анотация @Param

@Configuration

public class JavaConfig {

private Environment environment; - може да бъдат полета инджекнати през constructor

@Bean

public DataSource dataSource() {

DriverManagerDataSource driverManagerDataSource = new DriverManagerDataSource(); driverManagerDataSource.setDriverClassName(environment.getProperty("spring.datasource.driverClassName"));

driverManagerDataSource.setUrl(environment.getProperty("spring.datasource.url"));

driverManagerDataSource.setUsername(environment.getProperty("spring.datasource.  
username"));

driverManagerDataSource.setPassword(environment.getProperty("spring.datasource.  
password"));

return driverManagerDataSource;

* Example with BufferReader, който се инджектва през конструктор от Spring

@Configuration

Public class ApplicationBeanConfiguration {

@Bean

Public BufferReader bufferReader() {

Return new BufferReader(new InputStreamReader(System.in)

}

}

1. **Stored Procedures**

* Създава се процедура в SQL, името на променливата в метода трябва да е същото, както в процедурата.
* За процедури, които връщата резултата (@StoredNamedProcedure) в класа

@Procedure(“change\_book\_price\_by\_id”)

Void changeBookPriceById(Long book\_id);

1. **UUID primary key**

Private String id;

@Id

@GeneratedValue(generator = “uuid-str”)

@GenericGenerator(name = “uuid-str”,

strategy = “org.hibernate.id.UUIDGenerator”)

Public String getId() {

Return id;

}

1. **.reduce() – операции с елементи**

.reduce(Integer::sum)

1. **DateTimeFormatter**

* LocalDate може да се сравнява <, >, => и т.н дори когато формата на записа в таблицата е различен от подавания, само когато и двете са от един и същи тип (LocalDate).

LocalDate.*parse*(bufferedReader.readLine(), DateTimeFormatter.*ofPattern*("dd MMM yyyy"));

1. **SELECT DISTINCT**

* Селектира Users без да се повтарят,
* JOIN FETCH се използва за JOIN на таблици при Lazy, поставя се във заявката, вместо в Entity, за да не извлича цялата информация винаги, а само при изпълнение на Query-то



Еквивалент, но трябва да се използва fetchType = Eager:

