|  |  |  |
| --- | --- | --- |
| Annotation | Description |  |
| @Accessors  [Lombok] | Фокуси з методами getter та setter;  Див.:  Проект:jsLombokWithoutSpring, файл:DemoAccessors  Проект:jsBootDemo03, файл:InitiateUtils, ProviderEntity, FruitEntity |  |
| @AfterReturning(pointcut = "execution(\* prices())", returning = "catchedPrices") | Advice-метод в Aspect-классі, виконується лише після безпомилкового закінчення методу з основною логікою. Але до присвоєння результата цього методу якійсь змінній. Тому за допомогою @AfterReturning Advice можна змінювати повертаємий результат  Див.також:  Проект:jsSpringDemo07AOP, файл:Article1Aspect, Article2Aspect |  |
| @After(value = "execution(\* method(..))") | Advice-метод в Aspect-классі, який завжди виконується після закінчення методу з основною логікою. @After=@AfterReturning+@AfterThrowing, але на відміну від @AfterReturning та @AfterThrowing з допомогою анотації @After неможливо отримати доступ до виключення та до повертаємого результата.  Див.також:  Проект:jsSpringDemo07AOP, файл:Article1Aspect, Article2Aspect |  |
| @AfterThrowing(pointcut = "execution(\* method(..))", throwing = "myException") | Advice-метод в Aspect-классі, виконується після методу з основною логікою тільки якщо було виключення. Не впливає на перебіг програми при викидуванні виключення в методі основної логіки. Можна отримати доступ до самого виключення, але не обробити його.  Див.також:  Проект:jsSpringDemo07AOP, файл:Article1Aspect, Article2Aspect |  |
| @Around(value = "execution(\* method(..))") | Advice-метод в Aspect-классі, виконується до та після target-методу з основною логікою. З його допомогою можна: проводити якісь дії до та після роботи target-метода, отримати результат роботи target-метода та змінити його, виконати якісь дії, якщо з target-метода викидується виключення.  Див.також:  Проект:jsSpringDemo07AOP, файл:Article1Aspect, Article2Aspect |  |
| @AllArgsConstructor  [Lombok] | Створює один конструктор з усіма параметрами. Інших конструкторів немає, але якщо потрібно - можна дописати вручну. |  |
| @AllArgsConstructor(staticName = "of")  [Lombok] | Створює статичний фабричний метод з іменем of(), який повертає об'єкт створений з використанням **приватного** конструктора. Інших конструкторів немає, але якщо потрібно - можна дописати вручну. |  |
| @Autowired | Для методів, конструкторів та полів. Показує куди треба впровадити відповідний за класом або інтерфейсом бін (Біни шукає там, де є анотація @Component). Може впровадити залежність у приватне поле навіть, якщо немає конструктора чи сетера за допомогою Java Reflection API. Приклади:  Конструктор:  @Autowired  public MusicPlayer(Music music) { this.music = music; }  Поле:  @Autowired  private Music music;  Методи/Setters(на відміну від правил XML, для @Autowired назва методу може бути будь-якою):  @Autowired  public void setMusic(Music music) { this.music = music; }  @Autowired саме Spring анотація. Однак Spring фреймворк підтримує також стандартні JSR інструкції. Використання JSR анотацій дозволяє бути ще незалежнішим від Spring, а також перевикористовувати позначені класи в інших контейнерах, наприклад JBoss Steam. Аналог @Autowired в JSR анотаціях: @Inject (JSR 330). Різниця в тому, що @Autowired має параметр @Autowired(required=false). false – за відсутності біну буде встановлено null, true – викине Exception.  @Autowired Крім бінів можна впроваджувати і внутрішні об'єкти Spring:  - BeanFactory  - ApplicationContext  - ResourceLoader  - ApplicationEventPublisher  тобто. не потрібно буде реалізовувати Aware-інтерфейси, щоб отримати ці об'єкти.  Аналог @Qualifier("beanId") в JSR анотаціях: @Resource(name="beanId") (JSR 250).  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @Aspect | Aspect class відповідає за наскрізну функціональність: логування, перевірка прав(security check), обробка транзакцій, обробка винятків, кешування та інше).  Див.також:  Проект:jsSpringDemo07AOP, файл:Article1Aspect |  |
| @Bean | Для методів, які повертають об'єкти бінів. Метод має бути в классі позначенному @Configuration.  Див.також:  Проект:jsSpringDemo03Annotations1, файл:SpringConfig |  |
| @Before("execution(public int method01())") | Advice-метод в Aspect-классі, який виконується до методу з основною логікою.  Див.також:  Проект:jsSpringDemo07AOP, файл:Article1Aspect |  |
|  |  |  |
| @Builder  [Lombok] | Генерує методи, якими ми ініціалізуємо об'єкт по ланцюжку. Це зручно коли ми не хочемо використовувати (@AllArgsConstructor) конструктор з усіма параметрами (Якщо у нас клас незмінний, то в ньому єдиний конструктор з усіма параметрами).  Cat cat = Cat.builder().name("Tom").age(3).person(person).build(); |  |
| @Builder.Default  [Lombok] | Дозволяє задати значення за замовчуванням для певного поля в об'єкті, який будується за допомогою патерну Builder. |  |
| @Builder.ObtainVia(method = "...")  [Lombok] | Дозволяє вказати метод, який повинен бути використаний для отримання значення певного поля при використанні патерну Builder; |  |
| @Cleanup  [Lombok] | Автоматично закриває ресурси (наприклад, файли або з'єднання з базою даних), після виходу з блоку. |  |
| @Column | Для полів, методів/сеттерів/геттерів. |  |
| @CommonsLog  [Lombok] | Для генерації логгерів. |  |
| @Component  @Component("idName") | @Component=@Service=@Repository=@Controller - Одинакові і служать тілько для логіки developer-a(напр в MVC). Це аннотації для классів, з яких можливо створити бін.  Анотація @Component є загальною для всіх типів компонентів. Вона вказує на те, що клас є компонентом, який можна внедрити в інші класи або сконфігурувати за допомогою Spring.  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @ComponentScan("net.kiranatos")  @ComponentScan({"com.myPackage.first","com.myPackage.second"})  @ComponentScan(basePackageClasses = {A.class, B.class, C.class}) | Для классів. Ставиться в парі з @Configuration. Повідомляє Спрингу, класи якого пакету потрібно сканувати у пошуку бінів. Також можна вказати конкретні пакети для сканування, використовуючи параметр basePackages або basePackageClasses.  Див.також:  Проект:jsSpringDemo03Annotations1, файл:SpringConfig |  |
| @Configuration | Для классів. Вказує на клас, який містить налаштування бінів для контексту додатку.  Застосована до класу, який містить методи, що повертають об'єкти бінів або вказані конфігураційні параметри, що описують спосіб створення та конфігурування бінів: @Bean, @ComponentScan, @Import та інші, які також впливають на процес створення та налаштування бінів.  Див.також:  Проект:jsSpringDemo03Annotations1, файл:SpringConfig |  |
| @Controller | див. @Component  Анотація @Controller вказує на те, що клас є контролером, який відповідає за обробку запитів HTTP.  Якщо id не вказаний буде автоматично створений *id=poland* по імені класу class *Poland{}*.  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @Data  [Lombok] | Автоматично генерує *конструктор*, стандартні методи toString(), equals(), hashCode(), *getters* та *setters* для всіх полів класу, а також робить поля *private*. Може бути комбінованою з іншими аннотаціями: @NoArgsConstructor, @AllArgsConstructor, @Builder.  Іноді цю аннотацію краще замінити на @Setter та @Getter, якщо не потрібно переоприділять toString() та hashCode(), наприклад, з хеш-кодом можуть  виникати проблеми, якщо entity буде мати колекції або зв'язки з іншими *entity*.  Ще варіант: використання @ToString.Exclude та @EqualsAndHashCode.Exclude щоб не було циклічних методів.  Див.також:  Проект:jsLombokWithoutSpring, файл: DemoData |  |
| @Delegate  [Lombok] | Дозволяє делегувати реалізацію методів до іншого об'єкту; |  |
| @Email |  |  |
| @EnableAspectJAutoProxy | Вмикає автоматичне створення проксі-об'єктів для бінів, які позначені анотацією @Aspect. Це дозволяє виконувати аспектно-орієнтоване програмування(AOP). Коли @EnableAspectJAutoProxy застосовується до конфігураційного класу, Spring Framework створює автоматично проксі-об'єкти, які надають підтримку аспектів для бінів, позначених анотацією @Aspect.  Див.також:  Проект:jsSpringDemo07AOP |  |
| @Entity |  |  |
| @Enumerated |  |  |
| @EqualsAndHashCode  [Lombok] | Додає реалізації методів equals та hashCode |  |
| @EqualsAndHashCode.Exclude  [Lombok] | Дозволяє вказати, які поля не повинні братись до уваги при порівнянні об'єктів за допомогою методів equals() та hashCode(). Виключити поле у методах equals() та hashCode(). Якщо об'єкти мають двонаправлений зв'язок: посилаються один на одного, то використання методів toString, equals і hashCode (циклічний виклик методів вкладених об'єктів) призведе до помилки StackOverflowError. Див. також @ToString.Exclude. |  |
| @FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)  [Lombok] | Робить усі поля private і final |  |
| @FieldNameConstants  [Lombok] | Генерує клас з константами, які містять назви полів класу; |  |
| @GeneratedValue | **Автоматично** генерує значення ідентифікаторів для сутностей JPA. Вона може використовуватись без анотації @GenericGenerator, якщо використовувати стандартні генератори ідентифікаторів, такі як AUTO, IDENTITY, SEQUENCE або TABLE.  Якщо стратегія не вказана: вибереться AUTO -Hibernate сам призначатиме ID, ґрунтуючись насамперед на типі даних біля поля ID(наприклад, якщо ПК має UUID тип, то вибереться стратегія UUID).  Для SEQUENCE дивись також @SequenceGenerator  Для TABLE дивись також @TableGenerator  Див.також:  @GenericGenerator  @SequenceGenerator  Проект:jsBootDemo02, файл:FruitEntity  Проект:jsBootDemo14GeneratedID |  |
| @GenericGenerator | Задає **кастомний** генератор ідентифікаторів, який може бути використаний в поєднанні з будь-якою анотацією, що встановлює значення ідентифікатора, такою як @Id або @GeneratedValue.  Дозволяє налаштувати генератор ідентифікаторів за допомогою властивостей, таких як strategy, name, parameters. Властивість strategy вказує на стратегію генерації, наприклад, GenerationType.AUTO, GenerationType.IDENTITY, GenerationType.SEQUENCE тощо. Властивість name вказує на ім'я генератора, яке може використовуватись в анотації @GeneratedValue. Властивість parameters дозволяє передавати параметри до генератора ідентифікаторів.  @GeneratedValue(generator = "sequence-generator")  @GenericGenerator(name = "sequence-generator",  strategy = "org.hibernate.id.enhanced.SequenceStyleGenerator",  parameters = {  @Parameter(name = "sequence\_name", value = "user\_sequence"),  @Parameter(name = "initial\_value", value = "4"),  @Parameter(name = "increment\_size", value = "1") } )  @GeneratedValue(generator = "prod-generator")  @GenericGenerator(name = "prod-generator",  parameters = @Parameter(name = "prefix", value = "prod"),  strategy = "com.baeldung.hibernate.pojo.generator.MyGenerator")  Див.також:  @GeneratedValue  @SequenceGenerator  Проект:jsBootDemo02, файл:FruitEntity |  |
| @Getter  [Lombok] | Додає гетери для всіх параметрів класу |  |
| @Getter(lazy=true)  [Lombok] | Автоматично генерує метод getter, який ліниво ініціалізує поле, тобто значення поля обчислюється лише при першому виклику методу getter; |  |
| @Id | Може застосовуватись з @GenericGenerator, @GeneratedValue |  |
| @Import(SpringConfig2.class) | Для класів конфігурації, щоб імпортувати інші класи конфігурації. Її також можна писати над класами, що не є конфігураційними, але які містять компоненти, які потрібно включити в контекст додатку. |  |
| @JBossLog | Для генерації логгера. |  |
| @JoinColumn |  |  |
| @JoinTable |  |  |
| @Log  [Lombok] | Автоматично генерує екземпляр об'єкту логера. |  |
| @Log4j  [Lombok] | Для генерації логгерів. |  |
| @Log4j2  [Lombok] | Для генерації логгерів. |  |
| @ManyToMany | Див.також:  Проект:jjsBootDemo12EntityRelationship, файл: all classes in model directory  Chapter 1 in this file. |  |
| @ManyToOne | Див.також:  Проект:jjsBootDemo12EntityRelationship, файл: all classes in model directory  Chapter 1 in this file. |  |
| @NoArgsConstructor  [Lombok] | Додає конструктор без аргументів. Інших конструкторів немає, але якщо потрібно - можна дописати вручну. |  |
| @NoArgsConstructor(force = true)  [Lombok] | Генерує конструктор без параметрів, що встановлює всі поля в значення за замовчуванням, з вимогою їхньої ініціалізації (використовується тільки для немутабельних класів); |  |
| @NonNull  [Lombok] | Анотація, яка позначає, що поле не може бути null. |  |
| @NotBlank |  |  |
| @NotNull |  |  |
| @Order(-10) | Advice-методи одного Aspect-классу виконуються рандомно. Для того, щоб задати порядок виконання Advice-методів, потрібно рознести їх по різним класам. А в Aspect-классах проставити анотацію @Order(int) - чим меньше число, тим вище пріоритет.  Див.також:  Проект:jsSpringDemo07AOP, файл:Article2Aspect |  |
| @OneToMany | Див.також:  Проект:jjsBootDemo12EntityRelationship, файл: all classes in model directory  Chapter 1 in this file. |  |
| @OneToOne | Див.також:  Проект:jjsBootDemo12EntityRelationship, файл: all classes in model directory  Chapter 1 in this file. |  |
| @Parameter |  |  |
| @Pattern |  |  |
| @Pointcut("execution(\* show\*(int, ..))") | Pointcut - вираз, що описує де має бути застосований Advice-метод.  Див.також:  Проект:jsSpringDemo07AOP, файл:Article1Aspect, Article2Aspect |  |
| @PostConstruct | Вказує на метод, який повинен бути викликаний після того, як бін був повністю сконструйований та всі його залежності були встановлені. Такий метод використовується для виконання додаткової настройки біна або інших ініціалізаційних дій, які не можна виконати в конструкторі біна. Аннотація @PostConstruct може бути застосована до будь-якої кількості будь-яких методів в класі-біні.  Метод буде гарантовано викликаний тільки один раз після створення біна та перед тим, як бін буде використаний. Це забезпечує коректну ініціалізацію біна та його залежностей.  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @PreDestroy | Метод повинен бути викликаний перед знищенням біна. Коли контейнер Spring закінчує роботу з біном, метод буде викликаний для виконання завершальних дій, таких як звільнення ресурсів і закриття з'єднань з базами даних. Можна помітити кілька методів.  Для Prototype бінів не викликається destroy-метод! Виклики методу бінів з request, session, globalSession залежать від налаштувань, які використовуються для зберігання стану запиту, сесії та глобальної сесії. У випадку стандартних механізмів зберігання стану запиту та сесії, @PreDestroy метод не буде викликаний для бінів з цими областями видимості при закінченні запиту або завершенні сесії. Проте, для бінів з @Scope("globalSession"), @PreDestroy метод буде викликано при завершенні глобальної сесії. Якщо використовується налаштування для зберігання стану, які відрізняються від стандартних механізмів, поведінка може бути різною і може залежати від конкретної реалізації.  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @Profile | Дозволяє позначити бін, який буде використовуватися лише при заданому профілі додатку. |  |
| @PropertySource("classpath:myInfo.properties") |  |  |
| @Qualifier("beanId") | Для методів/сеттерів, конструкторів та полів. Вказуємо id бина, що хочемо впровадити. Використовується для конкретизації біна, якщо в контексті є більше одного біна з однаковим типом. Без @Autowired не вказується, бо тоді @Qualifier не матиме жодного впливу. Приклади:  *@Autowired*  *@Qualifier("nameIdObj")*  *private MyClass myClass;*  *@Autowired*  *public MyClassA(@Qualifier("id1") ClassB b, @Qualifier("id2") ClassC c){...}*  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @RequiredArgsConstructor  [Lombok] | Автоматично генерує конструктор з аргументами для всіх полів класу, які відмічені аннотацією @NonNull, final або обидвоми одночасно.  Якщо у класі є якийсь інший конструктор, необхідно додатково вказати @AllArgsConstructor або @NoArgsConstructor, щоб вказати який саме конструктор має використовуватись для генерації @RequiredArgsConstructor.  Див.також:  Проект:jsBootDemo03Lombok, файл: |  |
| @Repository | див. @Component  Анотація @Repository вказує на те, що клас є репозиторієм, який використовується для зберігання та взаємодії з даними. |  |
| @RestController | Використовується для створення веб-сервісу RESTful. Поєднує в собі аннотації @Controller та @ResponseBody. @Controller вказує, що клас є контролером, а @ResponseBody вказує, що повернене значення методу має бути сконвертовано в формат JSON або XML. |  |
| @Scope | Для класів, або над методом, який повертає об'єкт, який має стати біном. Використовується для вказівки області видимості (scope) компоненту (біна) в контейнері. Область видимості визначає, як довго компонент буде зберігатися в пам'яті та як він буде розподілятися між різними запитами.  @Scope можна використовувати зі звичайними класами, що не позначені жодною аннотацією, але бін не буде створений, потрібна додаткова відповіда аннотація(@Component, @Service, @Repository, @Controller, @RestController, та ін).  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @Scope("singleton") | Означає, що існує тільки один об'єкт біна на контейнер. Але це безглузда вказівка, оскільки біни є синглтонами за замовчуванням.  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @Scope("prototype") | Для кожного запиту з контейнера створюється новий об'єкт біна. |  |
| @Scope("request") | Бін створюється для кожного запиту веб-додатка. |  |
| @Scope("session") | Бін створюється для кожної сесії веб-додатка. |  |
| @Scope("globalSession") | Бін створюється для кожної глобальної сесії порталу.  Вказує, що бін повинен існувати на протязі життєвого циклу глобальної HTTP-сесії, якщо вона є використовується веб-застосунком, що підтримує сесії. В іншому випадку цей бін буде створений і збережений в області глобальної HTTP-сесії, незалежно від того, чи є сесія, або ні. Для використання цього типу області пам'яті біну, необхідно налаштувати відповідну функціональність у веб-застосунку, наприклад, за допомогою Spring Session. |  |
| @SequenceGenerator | Див.:  @GeneratedValue  Проект:jsBootDemo14GeneratedID, файл:FacultyEntity, FeedbackEntity, schema.sql |  |
| @Service | див. @Component.  Анотація @Service вказує на те, що клас є сервісом. Цей тип компонента використовується для виконання бізнес-логіки в додатку. |  |
| @Setter  [Lombok] | Додає сеттери для всіх параметрів класу |  |
| @Size |  |  |
| @Slf4j  [Lombok] | Автоматично генерує екземпляр об'єкту логера з бібліотеки SLF4J. |  |
| @SneakyThrows  [Lombok] | Робить виключення, що перевіряються, неперевіреними |  |
| @SpringBootApplication | Скорочена форма для комбінації інших трьох анотацій - @Configuration, @EnableAutoConfiguration і @ComponentScan. Вона вказує, що клас є головним класом конфігурації Spring Boot, який може бути запущений як Java програма. Анотація @Configuration вказує, що клас містить налаштування для контексту додатка Spring. @EnableAutoConfiguration вмикає автоматичну конфігурацію Spring Boot, а @ComponentScan дозволяє Spring знайти і підключити компоненти додатку. |  |
| @SuperBuilder  [Lombok] | Дозволяє наслідувати властивості патерну Builder у підкласах; |  |
| @Synchronized  [Lombok] | Автоматично додає синхронізацію на метод. |  |
| @Table |  |  |
| @TableGenerator | Див.:  @GeneratedValue  Проект:jsBootDemo14GeneratedID, файл:FruitEntity, PlanetEntity, PlantEntity, PlaneEntity |  |
| @Tolerate  [Lombok] | Дозволяє використовувати одночасно методи і конструктори створенні Lombok та написані вручну.  Докладніше дивись в:  Проект:jsLombokWithoutSpring, файл:DemoTolerate |  |
| @ToString  [Lombok] | Додає реалізацію методу toString |  |
| @ToString.Exclude  [Lombok] | Виключити поле у методі toString()  Див. також @EqualsAndHashCode.Exclude. |  |
| @ToString.Include  [Lombok] | Дозволяє вказати, які поля мають бути включені до результату виклику методу toString(); |  |
| @Transient |  |  |
| @Value  [Lombok] | Автоматично генерує *конструктор*, стандартні методи toString(), equals(), hashCode(), тільки *getters* для всіх полів класу, а також робить поля *private* i *final*. Може бути комбінованою з іншими. Подібний до @Data, але створює клас-значення (імутабельний клас з final полями та без генерації setter методів). |  |
| @Value(staticConstructor="of")  [Lombok] | Аналогічний до @Value, але з генерацією статичного методу-фабрики з назвою "of". |  |
| @Value.Check  [Lombok] | Дозволяє задати валідацію значень полів в іммутабельному класі; |  |
| @Value.Immutable  [Lombok] | Генерує immutable-клас, тобто клас, поля якого не можуть бути змінені після його створення; |  |
| @Value("${lang.name}")  @Value("1.2") | Для методів/сеттерів, конструкторів та полів. Використовується для інжекції простих значань або значень з конфігураційних properties-файлів. Приклади:  *public void setTitle(@Value("${app.anime}") String title) {...}*  *@Value("1.2")*  *private double square;*  *@Value("${events.file:target/events\_log.txt}")*  *private String filename;*  "events.file" - це ім'я змінної оточення, звідки ми візьмемо ім'я файлу.  "target/events\_log.txt" - це значення за замовчуванням, якщо змінна оточення events.file не буде встановлена.  Запис ${імяЗмінної:заЗамовчуванням} дозволяє отримати значення зі змінної оточення, або використати щось за замовчуванням.  Див.також:  Проект:jsSpringDemo02xmlAnnotations, файл:Language |  |
| @With  [Lombok] | Додає методи для кожного поля, які роблять клон незмінного об'єкта з одним зміненим полем. Cat anotherCat = cat.withName("Tom"); anotherCat - новий об'єкт у якого значення полів такі ж як у cat, крім поля name. |  |
| @Wither  [Lombok] | Генерує метод, який повертає копію об'єкту зі зміненим значенням одного з його полів; |  |
| @XSlf4j  [Lombok] | Для генерації логгера. |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Chapter 1. Entity Relationship in JPA/Hibernate/ORM**

The **@JoinColumn** annotation specifies the foreign key column in the owner table referencing the inverse table IDs. Useless if put in entity that don’t have joinColumn. Виключення: **@OneToMany –** тому-що сторона **Many** завжди є власником. З цієї ж причини у **@ManyToOne** немає параметру mappedBy().

|  |  |  |  |
| --- | --- | --- | --- |
| **@ManyToMany** | **@OneToMany** | **@OneToOne** | **@ManyToOne** |
| targetEntity() | | | |
| cascade() | | | |
| fetch() | | | |
| mappedBy() | | |  |
|  |  | optional() | |
|  | orphanRemoval() | |  |

**❄ ❄ ❄**

**Unidirectional** association - only one entity holds a reference to the other. Owner with annotation. Одностороннім називається відношення, власником якого є лише одна із двох сторін. Слід зазначити, що при цьому друга сторона про це нічого не знає. Hibernate вважатиме власником відносини ту сутність, в якій буде поставлена анотація відносини.

In a **@OneToOne** relationship, an instance of an entity is associated with only one instance of another entity.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class Car {  @OneToOne  @JoinColumn(name = "parkSpotId")  private ParkingSpot parkingSpot; } | class ParkingSpot { } |

H2 DB:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CAR\_ID** | **CAR\_NAME** | **PARK\_SPOT\_ID** |  | **PAR\_ID** | **PAR\_NAME** |
| 1 | Volvo | 2 | 1 | Los Santos |
| 2 | Mersedes | 1 | 2 | Paris |

Якщо видалити @JoinColumn(name = "parkSpotId"), то таблиці MY\_PARKING\_SPOT та MY\_CAR в БД не зміняться.

In a **@OneToMany** relationship, an entity has a reference to one or many instances of another entity.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class Department {  @OneToMany  @JoinColumn(name = "department\_id")  private List<Employee> employees; } | class Employee { } |

H2 DB:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DEP\_ID** | **DEP\_NAME** |  | **EMP\_ID** | **EMP\_NAME** | **DEP\_ID** |
| 1 | Epam | 1 | John | 2 |
| 2 | SoftServe | 2 | Sarah | 1 |

Якщо видалити @JoinColumn(name = "department\_id"), то таблиці з’явиться допоміжна таблиця.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DEP\_ID** | **DEP\_NAME** |  | **EMP\_ID** | **EMP\_NAME** |  | **DEP\_ID** | **EMP\_ID** |
| 1 | Epam | 1 | John | 1 | 2 |
| 2 | SoftServe | 2 | Sarah | 2 | 1 |

In a **@ManyToOne** relationship, many instances of an entity are associated with one instance of another entity.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class Student {  @ManyToOne  @JoinColumn(name = "school\_id", referencedColumnName = "schId")  private School school; } | class School {  @Id  @Column(name = "schId")  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Long id; } |

Параметр referencedColumnName дозволяє задати колонку явно, але й без нього Hibernate сам може підставити те що потрібно.

Якщо видалити @JoinColumn(name = "school\_id"), то таблиці MY\_SCHOOL та MY\_STUDENT в БД не зміняться.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STU\_ID** | **STU\_NAME** | **SCH\_ID** |  | **SCH\_ID** | **SCH\_NAME** |
| 1 | John | 2 | 1 | KPI |
| 2 | Sarah | 1 | 2 | KISI |

In a **@ManyToMany** relationship, many instances of an entity are associated with many instances of another entity.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class Book {  @ManyToMany  @JoinTable(name = "book\_author",  joinColumns = @JoinColumn(name = "book\_id"),  inverseJoinColumns = @JoinColumn(name = "author\_id"))  *private* Set<Author> authors; } | class Author { } |

H2 DB:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **BOO\_ID** | **BOO\_NAME** |  | **AUT\_ID** | **AUT\_NAME** |  | **BOOK\_ID** | **AUTHOR\_ID** |
| 1 | The Odyssey | 1 | J.K.Rowling | 1 | 2 |
| 2 | Harry Potter | 2 | Sarah | 2 | 1 |

Якщо видалити @JoinTable( … ), то таблиці MY\_BOOK, MY\_AUTHOR та BOOK\_AUTHOR в БД не зміняться.

**❄ ❄ ❄**

**Bidirectional** Associations is a relationship between two entities where each entity has a reference to the other.

Attribute *mappedBy* is in the **@OneToOne**, **@OneToMany** and **@ManyToMany** annotations, but NOT present in **@ManyToOne**.

The *mappedBy* attribute contains the name of the association-field on the owning side. We apply *mappedBy* attribute at the non-owning/inverse side of the relationship. The attribute instructs hibernate/JPA that don’t create extra column for this field in the table as it is already created at the owning side. In other words, when we use *mappedBy* attribute, it means that the relation between entities has already been mapped at owning side, so don’t do that twice. Additionally, *mappedBy* indicates that the relationship between entities is **bidirectional**. If we don’t use *mappedBy* attribute, it represents two **unidirectional** relationships. Further, there will be an additional mapping column in each direction.

It should be noted that in a **@OneToMany** relationship, the owner party may be only the **MANY** side, so the mappedBy attribute is only in the **@OneToMany** annotation.

In fact, **@ManyToOne** is always the owning side of a bidirectional association. Similarly, **@OneToMany** is always the inverse side of a bidirectional association.

In a **@OneToMany** bidirectional association, an entity has a reference to another entity. Additionally, the other entity has a collection of references to the first entity.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class User {  @ManyToOne  private Role **roleF**; } | class Role {  @OneToMany(mappedBy = “**roleF**”)  private User user; } |

When dealing with a **@ManyToMany** bidirectional association, it’s important to understand that each entity involved will have a collection of references to the other entity.

\* \* \*

**@OneToOne** from project jsBootDemo12EntityRelationship.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class Player {  @OneToOne  @JoinColumn(name = "roleId")  private Role role;  } | class Role {  @OneToOne(mappedBy = "role"/\*, cascade = CascadeType.ALL\*/)  private Player player;  } |

Пояснення про /\*, cascade = CascadeType.ALL\*/ дивись далі там де йдеться про помилку ***TransientObjectException***.

H2 DB:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PLA\_ID** | **PLA\_NAME** | **ROLE\_ID** |  | **ROL\_ID** | **ROL\_NAME** |
| 1 | John | 2 | 1 | Tank |
| 2 | Sarah | 1 | 2 | Wizard |

**@OneToMany** and **@ManyToOne** from project jsBootDemo12EntityRelationship.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class Planet {  @ManyToOne  @JoinColumn(name = "idSystem")  private StarSystem starSystemF;  } | class StarSystem { /\*The mappedBy attribute specifies the name of the attribute in the Planet entity that owns the relationship. In this case, the StarSystem entity doesn’t own the relationship, so we specify mappedBy = “starSystemF”.\*/  @OneToMany(mappedBy = "starSystemF")  private List<Planet> planets;  } |

H2 DB:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PLN\_ID** | **PLN\_NAME** | **ID\_SYSTEM** |  | **SYS\_ID** | **SYS\_NAME** |
| 1 | Sirius A II | 2 | 1 | Solar |
| 2 | Uran | 1 | 2 | Sirius |

**@ManyToMany** from project jsBootDemo12EntityRelationship.

|  |  |
| --- | --- |
| **Owning side** | **Inverse side** |
| class Lesson {  @ManyToMany  @JoinTable(name = "lesson\_pupil",  joinColumns = @JoinColumn(name = "les\_id"),  inverseJoinColumns = @JoinColumn(name = "pup\_id"))  private List<Pupil> pupilsG;  } | class Pupil {  @ManyToMany(mappedBy = "pupilsG")  private List<Lesson> lessons;  } |

H2 DB:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **LES\_ID** | **LES\_NAME** |  | **PUP\_ID** | **PUP\_NAME** |  | **LES\_ID** | **PUP\_ID** |
| 1 | Art | 1 | John | 1 | 2 |
| 2 | Music | 2 | Sarah | 1 | 1 |

**❄ ❄ ❄**

Помилка *TransientObjectException* та cascade = CascadeType.ALL.

В сутностях **@OneToOne** Role-Player в Util-класі InitiateUtilsRolePlayer при ініціалізації методом initForCascadeParametr() падала помилка TransientObjectException.

Рішення №1 : додати параметр в сутність Role в анотацію @OneToOne(mappedBy = "role", cascade = CascadeType.ALL).

З CascadeType в майбутньому потрібно буде ще експерементувати, глибоко я не копав.

Рішення №2 : був створенний інший метод init() зі спеціальним порядком додавання в БД і без параметру cascade в сутності. Моє припущення, чому у даному варіанті Transient Objects при Bidirectional зв’язку не падає: спочатку зберігаємо до БД сутність, яка не має FK колонки і їй ще не присвоїли посилання на іншу сутність. Хоча в такому випадку мав би падати з помилкою Lesson-Pupil. Також може бути причиною те, що першою сберігається ***Inverse Side***, а не власник. В цьому тверженні я поки-що не впевнений, але воно працює.