IŠ PRAEITOS PASKAITOS

- JpaRepository API yra `User[Entity]`
  - tik aplinkiniai sluoksniai, DB (tiksliau Entity Manager šiuo atveju) arba servisas, gali keistis duomenimis su šiuo API
  - taigi čia galima būtų bendrauti su serviso DTO objektais, tačiau kadangi mes šio sluoksnio neprogramuojam, tai nebūtinai
- pvz. jis gali išsaugoti User su id, firstname, lastname, username

IŠ PRAEITOS PASKAITOS

- Service API yra `User[Service]`
  - tik aplinkiniai sluoksniai, JpaRepository arba RestController, gali su juo keistis duomenimis
  - skirtas duomenis transformuoti ir perduoti iš DB arba atgal į DB
- jis gali sukurti User su daliniais duomenimis, pvz. gaudamas tik firstname+lastname
  - turi pasirūpinti ID generavimu (jei ne AUTO), ką rašyti į username laukelį, taip pat patikra ar User su tokiu firstname+lastname dar nėra, o jei yra ir firstname+lastname+username derinys unikalus, sugeneruoti kokį nors username

IŠ PRAEITOS PASKAITOS

- RestController API yra `User[Rest]`
  - tik aplinkiniai sluoksniai, Rest naudotojas (React) arba Service, gali su juo keistis duomenimis
  - skirtas surinkti duomenis iš Rest naudotojo įvairiais formatais ir perduoti į Service apdorojimui, o taip pat gautus duomenis iš Service paversti Rest naudotojui suprantamu formatu ir grąžinti
- jis gali turėti pvz `/create-user` ir leisti užklausą User kūrimui paduodant tik `{firstName, lastName}`
  - šiuo atveju `User[Rest]` bus tik su firstName ir lastName

IŠ PRAEITOS PASKAITOS

- Vėliau kai suprasite, kaip viskas veikia, galite sugalvoti, kaip sumažinti objektų skaičių, tačiau turite pasirūpinti, kad:
  - User[Entity] tiesiogiai niekada nenueitų Rest klientui
    - ir dėl saugumo, ir dėl to, kad klientas neturi prisirišti pvz. prie duomenų bazės vidinių savybių
  - neprašyti kliento User[Entity] duomenų
    - pvz. vienoj duomenų bazėj User[Entity] ID bus Long, kitoj String. Jei reikia unikaliai identifikuoti User, geriau tai daryti pvz. per username arba leisti UserService sugeneruoti ir išsaugoti nuo DB nepriklausantį unikalų ID

AKADEMIJA.IT

INFOBALT IR TECH CITY

JPA SĄRYŠIAI TARP ESYBIŲ. KOLEKCIJŲ SAUGOJIMAS. PAVELDĖJIMAS. KASKADINĖS OPERACIJOS.

Andrius Stašauskas

andrius@stasauskas.lt

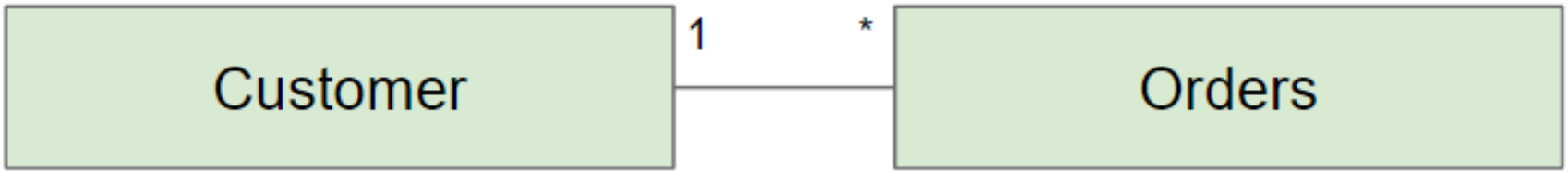
http://stasauskas.lt/itpro2018/

TURINYS

- Sąryšiai tarp esybių
- Kolekcijų saugojimas
- Kaskadinės operacijos
- Duomenų tikrinimas - validacija
- Entity paveldėjimas

SĄRYŠIAI TARP ESYBIŲ

SĄRYŠIŲ TARP ESYBIŲ VALDYMAS



- Saryšiai gali būti:
  - One-to-one
  - one-to-many
  - many-to-many
  - many-to-one
- Vienpusiai ir abipusiai
- Palaikantys Collection, Set, List ir Map tipus

SĄRYŠIAI: VIENAS-SU-VIENU

```
@Entity
public class Customer {
    @OneToOne
    private Credentials credentials;
}
```

- Turime nuorodą į vieną Credentials.

SĄRYŠIAI: VIENAS-SU-VIENU

```
@Entity
public class Customer {
 @Id
 public Long id;
 …
 @OneToOne
 public Credentials
 credentials;…
}
```

```
@Entity
public class Credentials {
 @Id
 public Long id;
 public String username;
 …
 @OneToOne(mappedBy="credentials")
 public Customer customer;
 …
}
```

- one-to-one sąryšis aprašomas @OneToOne anotacija
- Jei sąryšis dvipusis, ne savininko pusė turi naudoti mappedBy žymėjimą. Šiuo atveju Customer yra savininkas ir turės išorinį raktą į Credentials.

SĄRYŠIAI: VIENAS-SU-VIENU

```
@Entity
public class Customer {
    @OneToOne
    @JoinColumn(name="credentials_id")
    private Credentials credentials;
}
```

- Lentelėje Customer yra nuoroda: stulpelis credentials_id su foreign key į Credentials

SĄRYŠIAI: VIENAS-SU-DAUG

```
@Entity
class Cart {
                          @Id
       private Integer id;
                  @OneToMany
 private List<Good> goods;
}
```

```
@Entity
class Good {
 @Id
 private Integer id;
 @ManyToOne private
 Cart cart;
}
```

- many-to-one ryšys pažymimas @ManyToOne anotacija ir naudojamas esybėje, kurių bus "daug"
- one-to-many žymimas @OneToMany anotacija ir naudojamas klasėje, kurių bus tik "viena"
- Lentelėje Good yra stulpelis cart_id su FK į Cart

SĄRYŠIAI: VIENAS-SU-DAUG

```
@Entity
class Cart {
 @Id
 private Long id;
 @OneToMany(mappedBy="cart")
 private Set<Good> goods = new HashSet()

 public void addGood(Good good) {
     this.goods.add(good);
     good.setCart(this);
 }
}
```

```
@Entity
class Good {
 @Id
 private Long id;
 @ManyToOne
 @JoinColumn(name = "cart_id")
 private Cart cart;

 public Cart getCart() {
     return cart;
 }
}
```

- mappedBy elementas parodo, kad išorinį raktą turi kita esybė.

- Good lentelėje bus "cart" stulpelis - išorinis raktas į Cart lentelę.
- Good yra "savininkas", todėl kiekvieną kartą pridedant naują prekę į krepšelį, turi būti susiejamas krepšelis, kviečiant savininko susiejimo metodą setCart().

SĄRYŠIAI: DAUG-SU-DAUG

```
@Entity
class Cart {
 @Id
 private Integer id;
 @ManyToMany
 private List<Good> goods;
}
```

```
@Entity
class Good {
 @Id
 private Integer id;
 @ManyToMany
 private List<Cart> carts;
}
```

- Sąryšis žymimas @ManyToMany anotacija
- lentelės sujungtos trečia jungimo lentele Cart_Good

SĄRYŠIAI: DAUG-SU-DAUG

```
@Entity
class Cart {
 @Id
 private Integer id;
 @ManyToMany(mappedBy="carts
 private List<Good> goods;
}
```

```
@Entity
class Good {
 @Id
 private Integer id;
 @ManyToMany
 @JoinTable(table=@Table(name="G_C"),
 joinColumns=@JoinColumn(name="G_ID"),
 inverseJoinColumns=@JoinColumn(name="C_ID"))
 private List<Cart> carts;
}
```

- sąryšis realizuojamas naudojant asociatyvines lenteles (join table). Šiuo atveju lentelė "G_C"
- joinColumns nurodo esamos lentelės (Good) id stulpelį, inverseJoinColumns - susietos (Cart)
- jei sąryšis dvipusis, kita sąryšio pusė turi naudoti mappedBy elementą

KOLEKCIJŲ SAUGOJIMAS

JPA PERSIST MAP

- savininko pusėje reikalinga nurodyti @MapKey

```
@Entity
public class Author {
     @ManyToMany
     @JoinTable(
         name="AuthorBookGroup",
         joinColumns={@JoinColumn(name="fk_author",
             referencedColumnName="id")},
         inverseJoinColumns={@JoinColumn(name="fk_group",
             referencedColumnName="id")})
    @MapKey(name = "title")
    private Map<String, Book> books = new HashMap<String, Book>();
}
```

JPA PERSIST MAP

- priklausomai nuo key tipo:
  - `@MapKey` - unikalus atributas `Map#value` klasės lauke, jei raktas bazinio tipo
  - `@MapKeyEnumerated` - jei raktas `Enum` tipo
  - `@MapKeyTemporal` - jei raktas `Date` ar `Calendar` tipo
  - `@MapKeyJoinColumn(name="id_rakte")` - jei raktas sudėtingesnio `Class` tipo (raktas - objektas)
    - nesupainiokite: `@MapKeyColumn` - stulpelis su `@ElementCollection`
- raktai-laukai aišku turi būti pažymėti `@Temporal`, `@Enumerated`, raktai-klasės - `@Entity`

JPA PERSIST COLLECTION

- `Map<Long, Long>` ar `List<Long>`
  - galima tiesiog susikurti savo klasę `MyNumber1` ir/ar `MyNumber2`, viduje turėti `private Long` atributą
  - tuomet tai pavirs į `Map<Long, MyNumber1>` ar `Map<MyNumber1, MyNumber2>` ar `List<MyNumber1>`
- galima bus naudoti įprastas anotacijas `@OneToMany`, `@ManyToMany` ir t.t.
- daugiau kontrolės, nes yra atskiros lentelės ar laukai

JPA PERSIST COLLECTION

- arba naudoti `@ElementCollection` paprastiems tipams

```
@ElementCollection
private Collection<String> l = new ArrayList<>();
@ElementCollection
private Map<Long, Long> m = new HashMap<>();
```

  - bet norint dirbti su klasėmis, reiks žinoti `@Embeded`, `@Embeddable`, `@MapKeyColumn`, `@CollectionTable` ir kitas susijusias anotacijas
- mažiau kontrolės ir sunkiau keisti kodą (refactoring) ir migruoti duomenis

KASKADINĖS OPERACIJOS

KASKADINĖS OPERACIJOS

- jei nenurodyta kitaip, EntityManager operacijos taikomos tik tam entity objektui, kuris perduotas kaip parametras.
- Operacijos NEPERSIDUOS kitiems entity objektams, kurie pasiekiami per sąryšius.
- remove() operacijai tai dažnai ir yra norimas veikimas

```
class Cart {
  @OneToMany
  private List<Good> goods;
}
entityManager.remove(cart); // iš DB ištrins tik Cart
```

KASKADINĖS OPERACIJOS

- Jeigu turime dvi susietas esybes, turime jas išsaugoti
  - pradedant nuo savarankiškų ir baigiant „savininkais“

```
Customer customer = new Customer();
Credentials cred = new Credentials();
customer.setCredentials(cred);
em.persist(credentials);
em.persist(customer);
```

KASKADINĖS OPERACIJOS

- Tačiau galima saugojimo operacijas kaskaduoti susijusiems entity objektams, pvz.: persist() gali būti įvykdyta visoms susijusioms esybėms, nors kaip parametras perduodamas tik vienas entity objektas
- Reikia įsitikinti, kad susijusi esybė susieta (merged) prieš kviečiant persist(), t.y. ji negali būti detached

```
@Entity
public class Customer {
  @Id
  public Long id;
  …
  @OneToOne(cascade=CascadeType.Persist)
  public Credentials credentials;
  …
}
```

OPERACIJŲ TIPAI

- Galimi cascade tipai:
  - REMOVE - kaskadinis priklausančių objektų trynimas
  - PERSIST - priklausančių objektų išsaugojimas kartu
  - REFRESH - priklausančių objektų atnaujinimas
  - MERGE - objektų būsenos saugojimas ir užkrovimas
  - DETACH - objektų būsenos atsiejimas
  - ALL - visų operacijų kaskadinis vykdymas priklausantiems objektams
- Dvipusiams ryšiams reikia nurodyti kaskadas iš abiejų pusių, jeigu norima, kad jos veiktų simetriškai.

OPERACIJŲ TIPAI

- Dažnai yra naudojami MERGE ir DETACH tipai kartu

```
class User {
  @ManyToMany(cascade = {CascadeType.MERGE, CascadeType.DETACH})
  private List<Role> roles;
}
```

- tai reiškia, kad jei bus atsiejama User esybė, tai automatiškai bus atsietos ir Role esybės
- ir atvirkšiai: saugant User pakeitimus (merge) išsisaugos ir Role pakeitimai

SĄRYŠIŲ ELEMENTŲ ĮTERPIMAS

```
@Entity
class Cart {
 @Id
 private Long id;
 @OneToMany(mappedBy="cart",
  cascade = CascadeType.ALL)
 private Set<Good> goods = new HashSet();

 public void addGood(Good good) {
     this.goods.add(good);
     good.setCart(this);
 }
}
```

```
@Entity
class Good {
 @Id
 private Long id;
 @ManyToOne(cascade =
  {CascadeType.MERGE, CascadeType.DETACH})
 @JoinColumn(name = "cart_id")
 private Cart cart;

 public Cart getCart() {
     return cart;
 }
}
```

- atkreipkite dėmesį dar kartą: addGood metodas yra ne savininko pusėje tam, kad nustatytų ant norimo pridėti savininko.. save
- ManyToMany atveju galioja ta pati taisyklė
  - Cascade PERSIST netiks, nes atbulai kaskadinimas dažniausiai nevyksta jei egzistuoja savininkas-owner (bent jau su Hibernate)

SĄRYŠIŲ PAŠALINIMAS

- ManyToMany ar kitais atvejais, kai sąryšiams naudojama papildoma lentelė, ką reiškia REMOVE?
  - ar tai sąryšio pašalinimas ?
  - ar tai sąryšio ir susietos klasės objekto (lentelės įrašo) pašalinimas?
- @OneToOne ir @OneToMany turi orphanRemoval

> *Whether to apply the remove operation to entities that have been removed from the relationship and to cascade the remove operation to those entities.*

SĄRYŠIŲ PAŠALINIMAS

```
@OneToOne(orphanRemoval=true)
@OneToMany(orphanRemoval=true)
```

- kuris reiškia, kad bus trinami ne tik sąryšiai, bet ir surištas objektas, jei jis yra "našlaitis" (angl. orphan), t.y. prie nieko neprikabintas
  - ir jei jis nėra detached, nėra new, ir nėra removed

UŽDUOTIS #1 - ONETOONE

- sukurti prekei klasę ProductDetails papildomai informacijai saugoti
- ProductDetails turi turėti image ir description
  - prie produkto šių laukų turi nebelikti
- įgyvendinti OneToOne ryšį tarp Product ir ProductDetails
  - t.y. jei anksčiau turėjome product.getDescription() tai dabar bus product.getProductDetails().getDescription()
- turi pilnai veikti CRUD operacijos tiek produktui, tiek produkto papildomos informacijos esybei
- neturi kristi išimčių (exception)

UŽDUOTIS #1 - MANYTOMANY

- įgyvendinti ManyToMany ryšį tarp krepšelio ir prekių
- turi pilnai veikti CRUD operacijos tiek krepšeliui, tiek prekėms
- neturi kristi išimčių (exception)

ATIDĖTAS UŽKROVIMAS

ATIDĖTAS UŽKROVIMAS (LAZY LOADING)

- retai kada prireikia visų objekto sąryšių vienu metu
  - dažnai užtenka vieno ar poros sąryšių
- programos greitaveika gali būti optimizuojama nurodant, kad sąryšių užkrovimas yra atidėtas iki to momento, kol bus „paprašyta“
  - iškviestas getXxx() metodas
- Tai vadinama lazy loading
  - prisiminkime lazy loading beans

ATIDĖTAS UŽKROVIMAS (LAZY LOADING)

- Jeigu sąryšio fetch mode nėra nurodyta:
  - Vienos reikšmės sąryšiuose susijęs objektas užkraunamas iš karto, neatidedant.
  - Kolekcijos tipo sąryšiai pagal nutylėjimą – atidedami.
- Dvipusiuose sąryšiuose iš vienos pusės gali būti nustatytas atidėtas, iš kitos - momentinis užkrovimas
  - Normalu, kad poreikis gali skirtis, priklausomai iš krypties iš kurios daroma užklausa

ATIDĖTAS UŽKROVIMAS (LAZY LOADING)

- Atidėto užkrovimo direktyva `lazy` gali būti ignoruojama JPA implementacijos, nes iš principo paankstinus nebus pažeistas korektiškumas.
- Priešingai – išankstinio užkrovimo direktyva negali būti ignoruojama, nes sugriautų objekto korektiškumą, jeigu po užkrovimo entity pereitų į detached būseną – sąryšis taptų nebepasiekiamas.

ATIDĖTAS - LAZY

```
class Cart {
    @OneToMany(fetch=FetchType.LAZY)
    private List<Good> goods;
    public List<Good> getGoods() { return goods; }
}
```

- Goods iš DB bus atsiųstas tik iškvietus getGoods(...) metodą
- Jei metodas niekada nebus iškviestas - duomenys niekada nebus užkrauti iš DB

IŠ KARTO - EAGER

```
class Cart {
    @OneToMany(fetch=FetchType.EAGER)
    private List<Good> goods;
    public List<Good> getGoods() { return goods; }
}
```

- Duomenys užkraunami iš kart

ENTITY PAVELDĖJIMAS, VALIDAVIMAS, INDEKSAI IR KITA

DUOMENŲ TIKRINIMAS - VALIDACIJA

- Bean Validation – taip pat kaip ir RestController validavimo atveju:
  - `@NotNull`
  - `@DecimalMin ar @Digits`
  - `@Min ir @Max`
  - `@Size(min,max)`
  - `@Future ir @Past`
- Kitas anotacijas galima rasti JSR 380 standarte
- Įjungtas ne anotacija, o `application.properties` faile
  - nustatyti none norint išjungti (dokumentacija)

INDEKSŲ GENERAVIMAS

- indeksus dažniausiai naudojame:
  - jei nenorime tikrinti, ar jau egzistuoja kokių nors duomenų, kurie turi būti unikalūs (pvz. email)
  - jei norime, kad paieška ar rūšiavimas vyktų greičiau
- @Index anotacija leidžia apsirašyti ne-automatinius (PK/FK/Unique) indeksus
- @Index naudojama tik kaip kitų anotacijų sudedamoji dalis: Table, SecondaryTable, CollectionTable, JoinTable, TableGenerator

INDEKSŲ GENERAVIMAS

```
@Entity
@Table(name = "region", indexes = {
    @Index(name = "idx_code",  columnList="code", unique = true),
    @Index(name = "idx_name", columnList="name ASC", unique = false),
    @Index(name = "idx_name_code", columnList="name,code DESC")
})
public class Region{
    @Column(name = "code", nullable = false)
    private String code;
    @Column(name = "name", nullable = false)
    private String name;
}
```

DUOMENŲ PATIKRINIMAS AR INDEKSAI?

- Jeigu naudojate pvz. `existsBy` duomenų patikrai prieš kuriant įrašą, pirma jūs galite patikrinti, o tuo metu kas nors gali įdėti toki patį įrašą dar prieš jums spėjant įvykdyti transakciją
- Todėl norint užtikrinti, kad laukas bus unikalus, vis tiek reikia prie lentelės sukurti `@Index` unikalų indeksą

```
@Table(name = "user", indexes = {
  @Index(name = "idx_nickname",
         columnList = "nickname",
         unique = true) } )
```

JPA LIFECYCLE METODAI

- Specialios anotacijos entity metodams pažymėti, jeigu jie turi būti iškviečiami tam tikru momentu gyvavimo cikle
  - @PrePersist
  - @PreRemove
  - @PostPersist
  - @PostRemove
  - @PreUpdate
  - @PostUpdate
  - @PostLoad
- daugiau informacijos specifikacijoje ir čia

JPA LIFECYCLE METODAI

- turėdami faktūrą ir kliento duomenis joje, galime automatiškai sukurti klientą, tiesiog saugodami faktūrą

```
@Entity class Invoice {
    private Customer cust;
    private String name;
    private Address address;
    @PreCreate
    public void onPreCreate() {
        // Automatically create a new customer
        if (getCustomer() == null) {
            Customer cust = new Customer();
            cust.setName(getName());
            cust.setAddress(getAddress());
            cust = em.merge(cust); // attach this new entity
            setCustomer(cust); // and here we change a relationship
        }
    }
}
```

ENTITY PAVELDĖJIMAS - TĖVAS

- kai norime aprašyti bendras savybes grupei Entity klasių, tačiau bazinės klasės nenorime turėti kaip atskiros entity:

```
@MappedSuperclass
public class Person {
    @Id
    protected Long id;
    protected String name;
    @Embedded
    protected Address address;
}
@Entity
public class Customer extends Person {
    @Transient
    protected int orderCount;
    @OneToMany
    protected Set<Order> orders = new HashSet();
}
```

ENTITY PAVELDĖJIMAS - VIENOJE LENTELĖJE

- kai norime aprašyti bendras savybes grupei Entity klasių, tačiau bazinės klasės nenorime turėti kaip atskiros entity:

```
@Entity
@Table(name = "VEHICLE")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "VEHICLE_TYPE")
public abstract class Vehicle { /* Vehicle class code */ }
@MappedSuperclass public abstract class PassengerV extends Vehicle {}
@Entity
@DiscriminatorValue(value = "Bike")
public class Bike extends PassengerV { }
@Entity
@DiscriminatorValue(value = "Car")
public class Car extends PassengerV { }
```

- daugiau informacijos čia ir čia

PAPILDOMOS GALIMYBĖS

- Rakinimas (Locking) yra techika norint suvaldyti situacijas, kai tuos pačius duomenis vienu metu keičia keli naudotojai. Užrakinant duomenis užtikriname, kad tik vienas iš naudotojų vienu metu galės keisti duomenis.
  - JPA palaiko pesimistinį ir optimistinį režimus
    - kontroliuojama su anotacijomis @Version ir @LockModeType
    - Optimistinis: visiem leidžiama skaityti ir keisti, prieš commit įvyksta patikrinimas, ar įrašo versija nepasikeitė. Jei pasikeitė – metamas exception. Geresnė greitaveika.
    - Pesimistinis: užrakinama DB įrašo lygyje ir neleidžiama rašyti ir/arba skaityti.
- Trys būdai, kaip atlikti rakinimą:
  - EntityManager methodai: lock, find, refresh
  - Query methodas: setLockMode
  - NamedQuery anotacija: lockMode elementas

KITA, KAS SVARBU

- kaip spręsti problemas pesimistic/optimistic lock
- Criteria API
- JPQL JOIN TREAT
- reikšmių konvertavimas su @Convert
- @NamedEntityGraph
- @ConstructorResult
- Java klasių generavimas iš SQL ir DB lentelių

NUORODOS

- JPA 2.1 Specification PDF
- EE7/Persistence Tutorial
- JPQL pavyzdžiai čia ir čia
- JPQL sintaksė
- Vidinės užklausos ir Criteria
- NamedEntityGraph čia ir čia
- Dinamiškai formuojamos vardinės užklausos
- JSR-338 Java Persistence 2.1 Oficiali dokumentacija
- JPA 2.x specification
- Hibernate Community Documentation, EntityManager

UŽDUOTIS #2 - PAVELDĖJIMAS

- prekę pakeisti į vienos lentelės paveldimumo tipo esybę su diskriminatoriumi
- praplėsti prekę keliomis prekių grupėmis, pvz. Clothes ir Toys
- įsitikinti, kad tokias prekes iš lentelės įmanoma nuskaityti
  - kad jas įmanoma išsaugoti
  - kad jas galima pridėti ir išimti iš krepšelio

KITOJE PASKAITOJE

Spring Security. Live coding