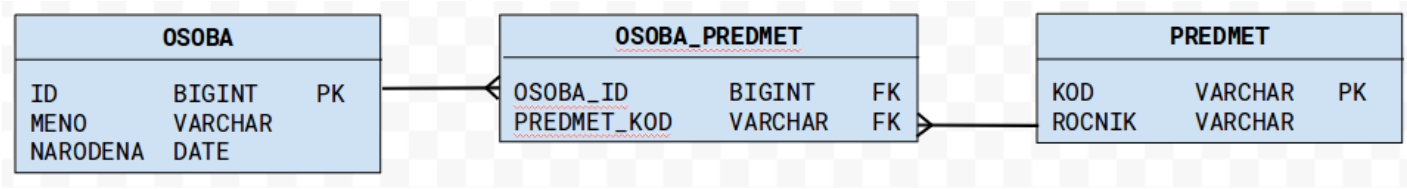
Implementačná úloha 1.

Na obrázku je zobrazený diagram databázy s údajmi o osobách a predmetoch, ktoré učia.



V zip-súbore máte pripravený netbeans-projekt **zapocet2** pre vývoj aplikácie pracujúcej s touto databázou.

Projekt obsahuje (v balíku zapocet2) tri zdrojové súbory:

- **Osoba.java**: Dátová trieda pre prácu s osobami. Obsahuje zatiaľ len deklarácie dátových členov (a tiež metódy hashCode, equals, a toString)
- **Predmet.java**: Dátová trieda pre prácu s predmetmi. Obsahuje zatiaľ len deklarácie dátových členov (a tiež metódy hashCode, equals, a toString)
- **Zapocet2.java**: Obsahuje deklarácie troch metód pre prácu s uvedenou databázou.
  - Set<Osoba> **vyucujuci**(EntityManager em, String kodPredmetu)
    Vyhľadá v DB všetkých vyučujúcich daného predmetu.
  - int **pocetPredmetov**(EntityManager em, String meno)
    Zistí počet predmetov, ktoré učí vyučujúci.
  - boolean **pridajVyucujuceho**(EntityManager em, String meno, String kodPredmetu)
    Pridá predmetu ďalšieho vyučujúceho.

  Vašou úlohou je implementácia týchto metód podľa špecifikácie. Presnú špecifikáciu týchto metód nájdete v komentáry **v hlavičke** každej z nich.

  Okrem týchto metód obsahuje súbor aj **main** funkciu. Funkcia main je už implementovaná. Slúži len pre vás na testovanie metód. Jej implementáciu si môžete zmeniť podľa potreby. Neodovzdáva sa.

Pokyny k implementácii.

- Aby ste mohli metódy pracujúce s databázou implementovať je nutné najprv dokončiť implementáciu tried **Osoba** a **Predmet.** Existujúci kód v súboroch nie je potrebné modifikovať (názvy a typy dátových členov nemeniť!) Stačí ho doplniť o:
  - **gettre a settre** pre dátové členy
  - potrebné **JPA anotácie**, tak aby sa entitné triedy mapovali na databázové tabuľky s presne takou štruktúrou ako je uvedená v diagrame hore a pritom:
    - kľúč ID tabuľky OSOBA bude **autogenerovaný**
    - kľúč KOD tabuľky PREDMET nie je autogenerovaný.
    - Názov prepojovacej tabuľky (OSOBA_PREDMET) aj názvy jej stĺpcov musia byť tiež také ako je uvedené v diagrame.
- Pracujte s lokálnou derby/sample databázou. Odporúčam postupovať tak, že najprv dokončíte implementáciu entitných tried a z nich si necháte vygenerovať tabuľky .
  Dôležité je aby vygenerované tabuľky mali **presne tú istú štruktúru** ako je uvedená v **diagrame** hore. (Názvy tabuliek, stĺpcov a ich typy musia byť zachované).
  Súbor **persistence.xml** konfiguráciou pripojenia na derby datbázu už máte v projekte tiež pripravený.
  Poznámka. Opačný postup, t.j. generovať entitné triedy z DB, je možný ale neodporúčam ho.

Pokyny pre odovzdanie:

Do miesta odovzdania **t1-u1db** *sa odovzdávajú sa zdrojové súbory:*

- **Osoba.java**
- **Predmet.java**
- **Zapocet2.java**

samostatne, t.j. **NEZOZIPOVANÉ!**

Implementačná úloha 2.

Implementujte RESTful servis, ktorý bude sprístupňovať informácie skúškach zo študijných predmetov. Informácie o skúške z konkrétneho predmetu predstavujú resource, ktorého stav reprezentovaný vo formáte XML má štruktúru:

```
<?xml version="1.0" encoding="UTF-8"?>
<skuska>
    <predmet>OOP</predmet>
    <den>utorok</den>
    <student>Jozef Mrkvicka</student>
    <student>Janko Hrasko</student>
</skuska>
```

kde:

- element **predmet** obsahuje skratku predmetu, ktorá je zároveň jednoznačným identifikátorom skúšky. Teda musí byť zadaná.
- element **den** udáva **deň** konania skúšky.
- elementy **student** obsahujú mená študentov prihlásených na skúšku.

Pozn. Uvedený dokument je ukážka. XML dokumenty, s ktorými servis pracuje budú mať elementy s rovnakými názvami a štruktúrou, odlišovať sa budú len textovým obsahom a počtom elementov. (Elementy den a student nemusia byť zadané, študentov môže byť aj viac. Poradie elementov nie je dôležité.)

POZOR dôležité!

- Vytvorte **java web application** projekt a nazvite ho **RestApp**
- Package pre resource nazvite: **rest.** V package **rest** vytvárajte aj všetky ostatné (dátové) triedy.
- Path – koreňový resource nazvite: **skuska**

  Pozn. Pre anotáciu @Singleton importujte **javax.inject.Singleton**
  Nezabudnite na anotáciu @XmlRootElement

Servis inicializujte tak, že po štarte (deploymente) bude vytvorený jediný resource a to pre skúšku z predmetu **`OOP`**, pričom deň konania skúšky bude **`utorok`** a prihlásení nebudú žiadni študenti. Resoursy pre skúšky z ďalších predmetov bude možné pridávať pomocou metódy POST.

1. Pre koreňový resource **skuska** implementujete metódu
POST, ktorá slúži pre vytvorenie nového resoursu s informáciami o skúške.

- akceptuje XML dokument (MIME: APPLICATION_XML) podľa horeuvedenej špecifikácie.
- a vracia reťazec (MIME: TEXT_PLAIN) so skratkou predmetu, prevzatou z elementu predmet.

Pozor. Metóda musí najprv preveriť, či resource s danou skratkou predmetu už neexistuje. Ak exsituje, neurobí nič a vráti reťazec **`CHYBA`**.

2. Všetky informácie o skúške pre konkrétny predmet sú prístupné ako subresource s URI
skuska/{predmet}, kde reťazec predmet je skratka predmetu.
Pre tento resource implementujte nasledujúce metódy:
GET pre MIME TEXT_PLAIN**:** vráti počet študentov prihlásených na skúšku. Ak resource neexistuje, malo by sa vrátiť **0** prípadne HTTP 204 alebo HTTP 404 .
GET pre MIME APPLICATION_XML: vyhľadá skúšku podľa skratky predmetu zadaného v URL a vráti informácie ako XML dokument s horeuvedenou štrukrúrou. Ak resource pre skúšku z daného predmet neexistuje, malo by sa vrátiť HTTP 204 alebo 404 *(Pomôcka, implementacia vráti jednoducho null)*
POST slúži na prihlásenie študenta na skúšku. Očakáva reťazec s menom študenta (MIME: TEXT_PLAIN). Ak je študent s daným menom ešte nie je prihlásený, pridá ho medzi prihlásených študentov. Metóda nevracia nič. Ak je už študent prihlásený nerobí nič.

3. Servis umožňuje tiež zistiť skúšky, na ktoré je študent prihlásený. Túto funkcionalitu implementujte pomocou metódy **GET** pre koreňový resource **skuska**. Meno študenta je zadané ako parameter požiadavky **student**

(`@QueryParam("student")` - pozri ťahák). Metóda zistí, na ktoré skúšky je študent prihlásený a vráti reťazec, zložený zo skratiek predmetov, na ktoré je prihlásený, oddelených medzerou. (MIME: TEXT_PLAIN) Ak meno študenta nie je zadané ako parameter požiadavky nevráti nič, príp. prázdny reťazec.

Pokyny pre odovzdanie:
Odovzdávajú sa **všetky vytvorené zdrojové súbory,** t.j. všetky súbory z adresára **src/java/rest.**
Súbory sa odovzdávajú **nezozipované** do miesta odovzdania **t1-u2rest .**

Implementačná úloha 3.

Implementujte jednoduchú klientskú aplikáciu pre skúškový RESTful servis zadaný v **úlohe 2.**

POZOR dôležité!

- Vytvorte **java application** projekt a nazvyte ho **RestClient.**
  Pozn. Wizard vám automaticky vytvorí package **restclient** a v ňom triedu **RestClient** s hlavnou **main** metódou**.**
- Do package **restclient** budete dávať aj všetky ďalšie zdrojové súbory (t.j. jersey-client aj dátovú triedu)

Implementujte klientskú aplikáciu (metódu main), ktorá prostredníctvom vygenerovaného jersey-clienta volá metódy skúškového REST servisu a:

1. zistí, v ktorý **deň** je skúška z predmetu **OOP** a vypíše deň na štandardný výstup.
2. následne **prihlási jedného študenta** na skúšku z **OOP**. Ako meno študenta zadajte **vaše ID-študenta.**
3. nakoniec zistí **počet prihlásených študentov** na predmet **OOP** a vypíše ich na štandardný výstup.

Poznámka: Ak teda implementujete servis z úlohy 2 správne podľa špecifikácie, tak po jeho naštartovaní by táto klientská aplikácia mala na obrazovku vypísať:

```
utorok
1
```

Pokyny pre odovzdanie:
Odovzdávajú sa **všetky vytvorené zdrojové súbory,** t.j. všetky súbory z adresára **src/restclient.**
Súbory sa odovzdávajú **nezozipované** do miesta odovzdania **t1-u3klient .**