Medicinska dijagnostika – JPA

1. Uvod

Projekat predstavlja sistem za realizaciju Medicinske dijagnostike koga čine podaci o simptomima, bolestima, pacijentima, njihovim dijagnozama i terapijama sa prepisanim lekovima. Dijagnoze se ne mogu kreirati dok se ne zavedu podaci o pacijentu, kao i simptomima i bolestima. Takođe terapije ne mogu postojati dok se ne uradi bar jedna dijagnoza nekom pacijentu i dokle god nema lekova za određenu terapiju.

Pravila nalažu da se jedan simptom može naći kod više bolesti, kao i da jedna bolest može imati više simptoma, pa iz tog razloga imamo međuentitet za simptome i bolesti.

Takođe jedan pacijent može imati više dijagnoza i za jednu dijagnozu može se odrediti više terapija.

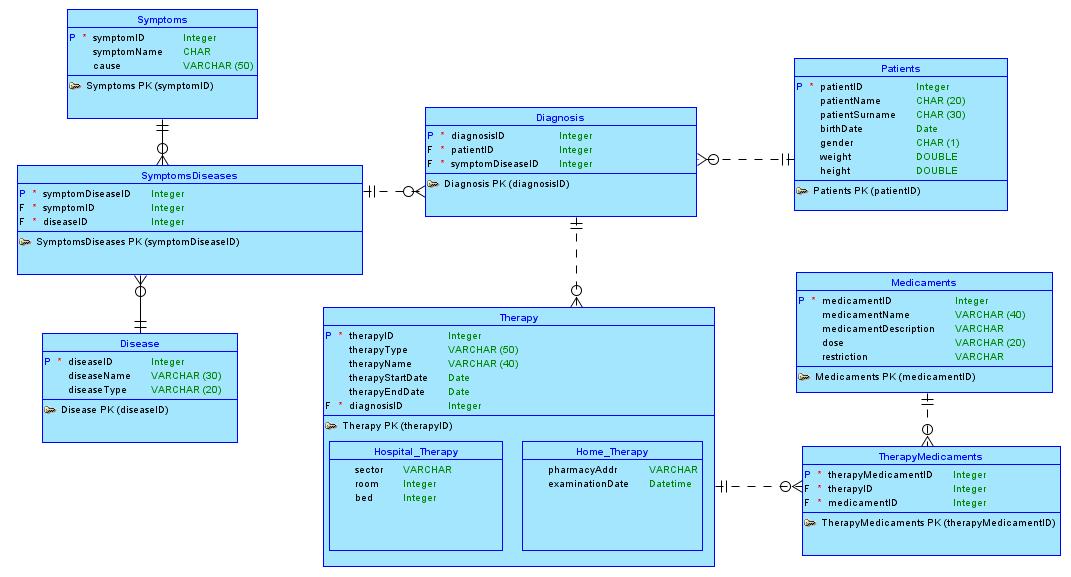
Jedna terapija može uključiti više lekova, takođe se i jedan lek mođe uključiti u više terapija.

1. ERD Dijagram

Na dijagramu ispod prikazan je Entity-Relationship Diagram gde se može videti logički model sistema za Medicinsku dijagnostiku. Veza M:N između entiteta Bolesti - Simptoma, Terapija - Lekova je rešena uvođenjem međuentiteta kako bi vezu M:N transformisali u dve veze 1:N i M:1. Imamo prikaz generalizacije kod eniteta Terapija koja je generalizovana za podentitete Kućna terapija i Bolnička terapija. Ova generalizacija će biti rešena SINGLE\_TABLE metodom gde su u fizičkom modelu svi podaci u jednoj tabeli. U logičkom kao i u OOP modelu imamo dve podklase koje nasleđuju osobine glavne apstraktne klase i sadrže samo svoje specijalne atribute.

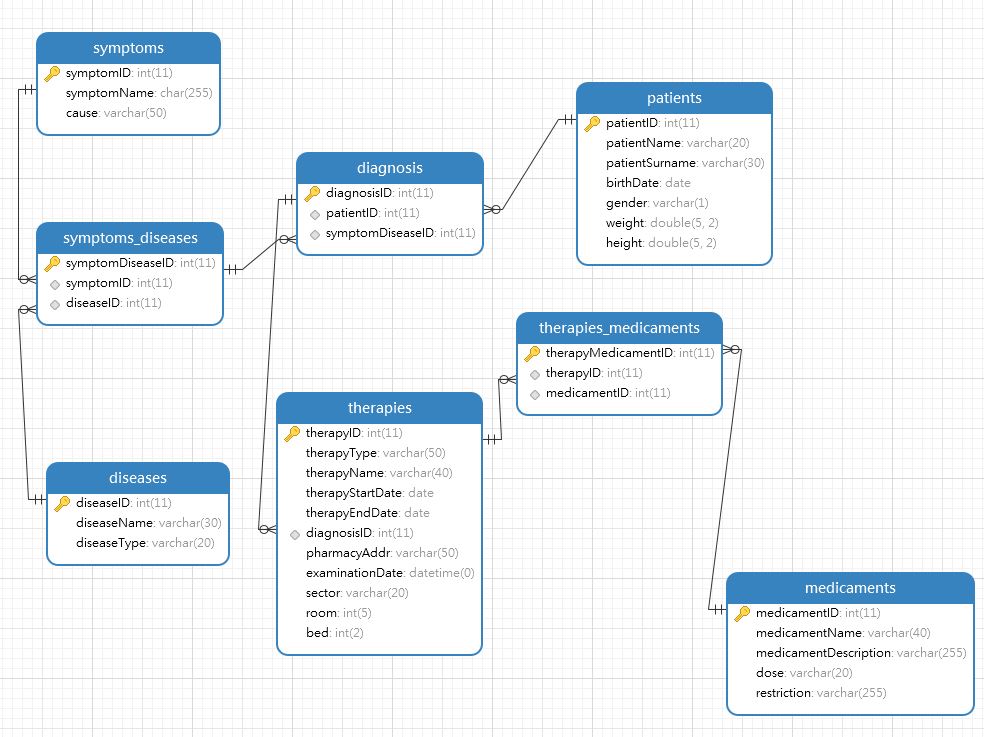
Mapiranje veza u entitetu Terapija je rešena u glavnom entitetu. A diskriminator za specijalizaciju će prema svojoj vrednosti odrediti kog je specijalnog tipa konkretna Terapija.

Kada se reše veze M:N, vidimo da su sve veze na dijagramu 1:N tipa zbog navedenih poslovnih pravila.

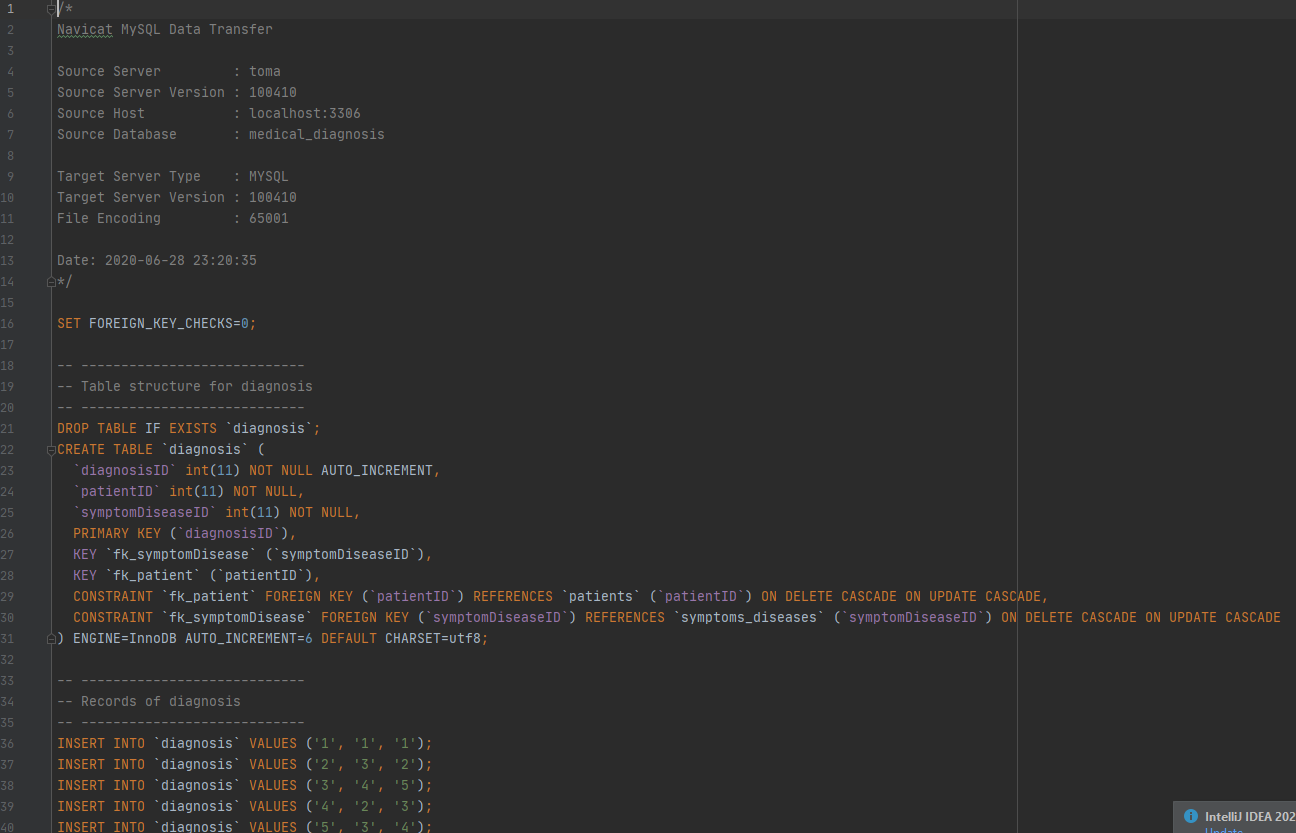


1. Fizički model

Na modelu ispod vidimo prikaz fizičkog modela baze gde su entiteti prevedeni u tabele i koji se ne razlikuje mnogo od logičkog modela. Jedina bitna razlika je što je generalizacija vezana za entitet Terapija rešena **SINGLE\_TABLE** metodom gde imamo jednu tabelu u kojoj su **nullable** polja oba entiteta kao i DiscriminatorColumn – therapyType koji može imati vrednosti Hospital ili Home kako bi se znalo koja će dodatna polja biti popunjena a koja imati vrednost null.



Ispod je prikaz sadržaja sql skripte kojim se generiše fizički model baze podataka.



1. Podešavanje razvojnog okruženja

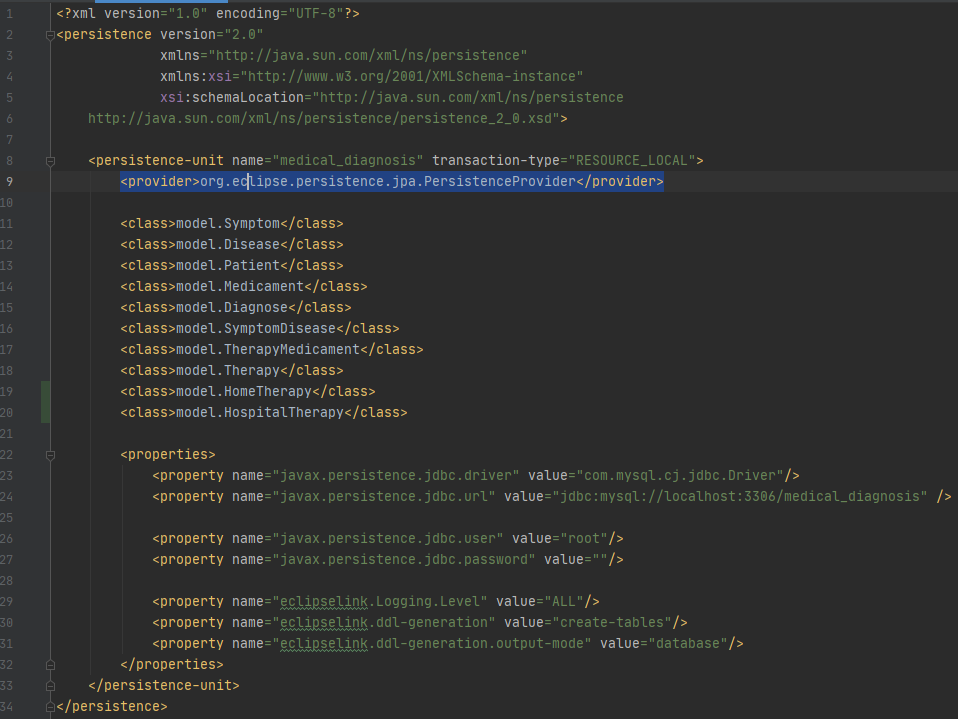
* DBMS koji se koristi je MySQL gde je kreirana šema medical\_diagnosis i koja je povezana sa JPA projektom preko .jar biblioteke konektora **mysql-connector-java-8.0.18.jar.**
* Provajder je Eclipse koji je preuzet preko Maven-a i naveden je u persistence.xml fajlu pod tagom

**<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>**

* Za razvoj projekta se koristi **IDEA IntelliJ 2020** okruženje.
* Potrebno je bilo setovati SDK za Java 14 verziju, kao i biblioteke potrebne za JPA poput javax.persistence, mysql-connector, org.eclipse.persistence.jpa i ostale. Takođe unutar src direktorijuma je bilo potrebno napraviti META-INF folder gde se nalazi persistence.xml kao JPA konfiguracuioni fajl. Takođe je bilo potrebno napraviti model folder sa svim definisanim klasama kao JPA entitetima i ui folder gde se nalazi glavna upravljačka klasa sa kojom se može vršiti interakcija preko konzole.

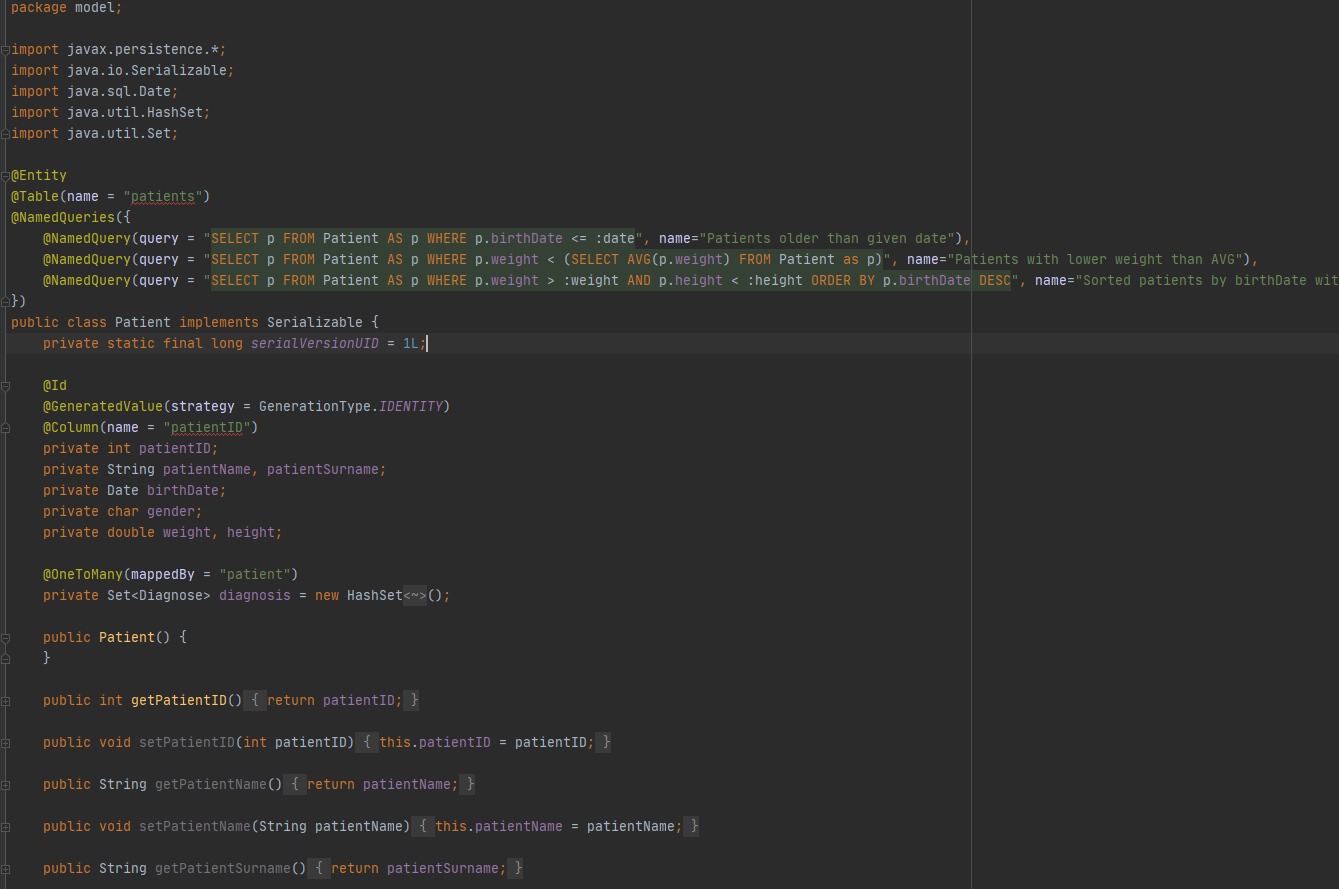
1. Klase sa relacijama i persistence.xml fajl

Persistence.xml fajl je prikazan ispod sa definisanim provajderom, klasama i jdbc paramaterima.

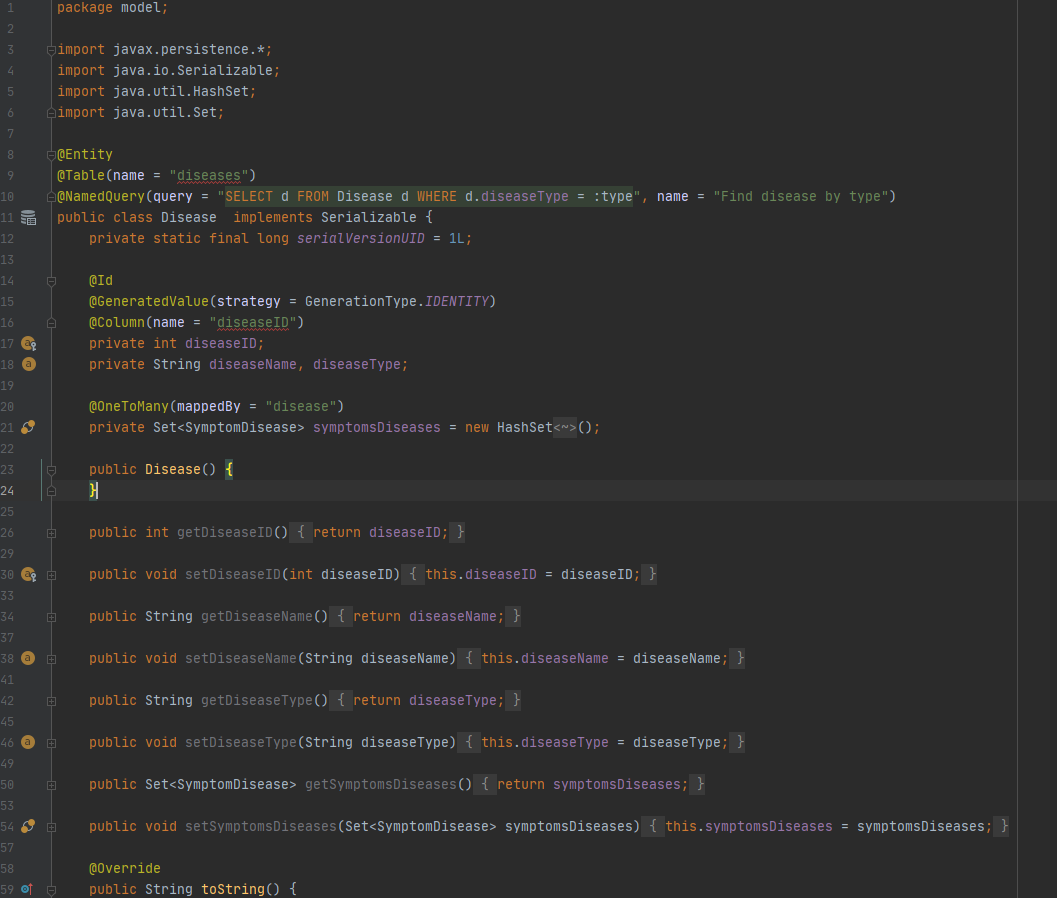


Prikaz klase Pacijent je ispod. Ona je definisana kao JPA perzistentan entitet i povezana je sa tabelom patients. Takođe vidimo nekoliko Named upita definisanih iznad klase.

Svaka klasa kao enitet mora da implementira interfejs Serializable jer se u Entity Manager vrši serijalizacija entiteta kako bi se čuvale instance. Za id entiteta smo stavili da je GenerationType. IDENTITY što znači da korisitimo AUTO\_INCREMENT mehanizam generisanja vrednosti primarnog ključa. Tako je i za sve ostale entitete definisano.



Prikaz klase Disease koja takođe sadrži jedan Named upit i sve ostalo je na isti način definisano. Takođe imamo i mapiranu relaciju 1:N sa međuentitetom symptomsDiseases koja kao atribut ima HashSet strukturu podataka za brz pristup.



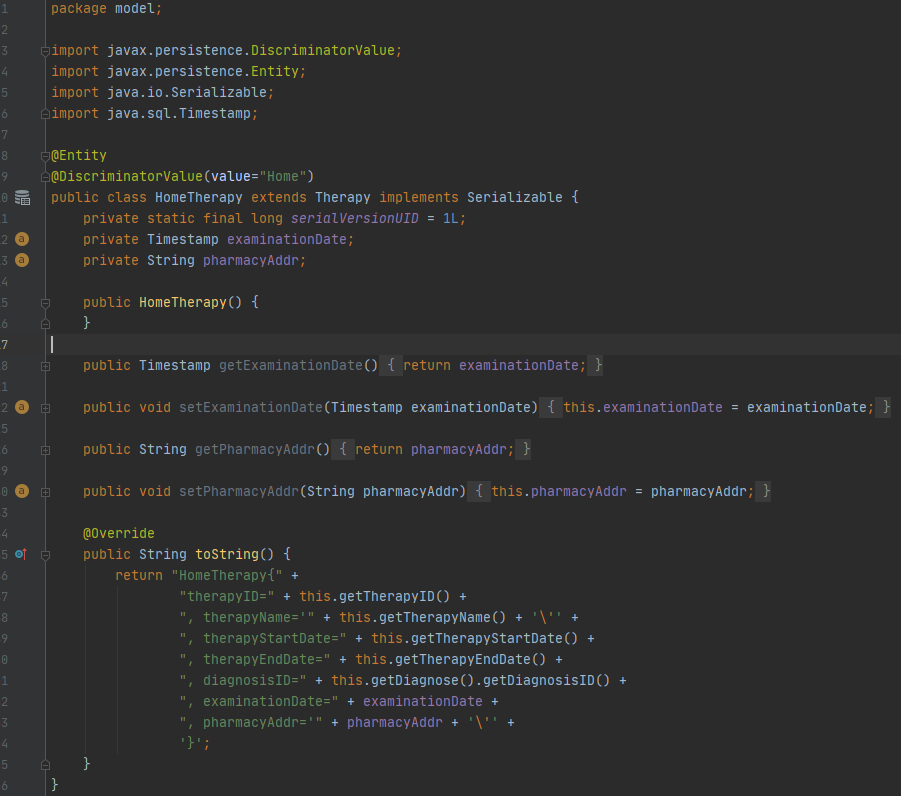
Prikaz **Therapy** klase koja je apstraktna i generalizacija za klase HomeTherapy i HospitalTherapy.

Strategija generalizacije je **SINGLE\_TABLE** i korisit se therapyType kao diskriminatorska kolona. Takođe tabela ima vezu N:1 sa Dijagnozama jer se više terapija može naći u jendoj dijagnozi. S druge strane ima vezu 1:N sa međuentitetom sa medikamentima jer se više lekova može koristiti za jednu terapiju i zato se ovde takođe koristi HashSet **therapyMedicament** objekata.



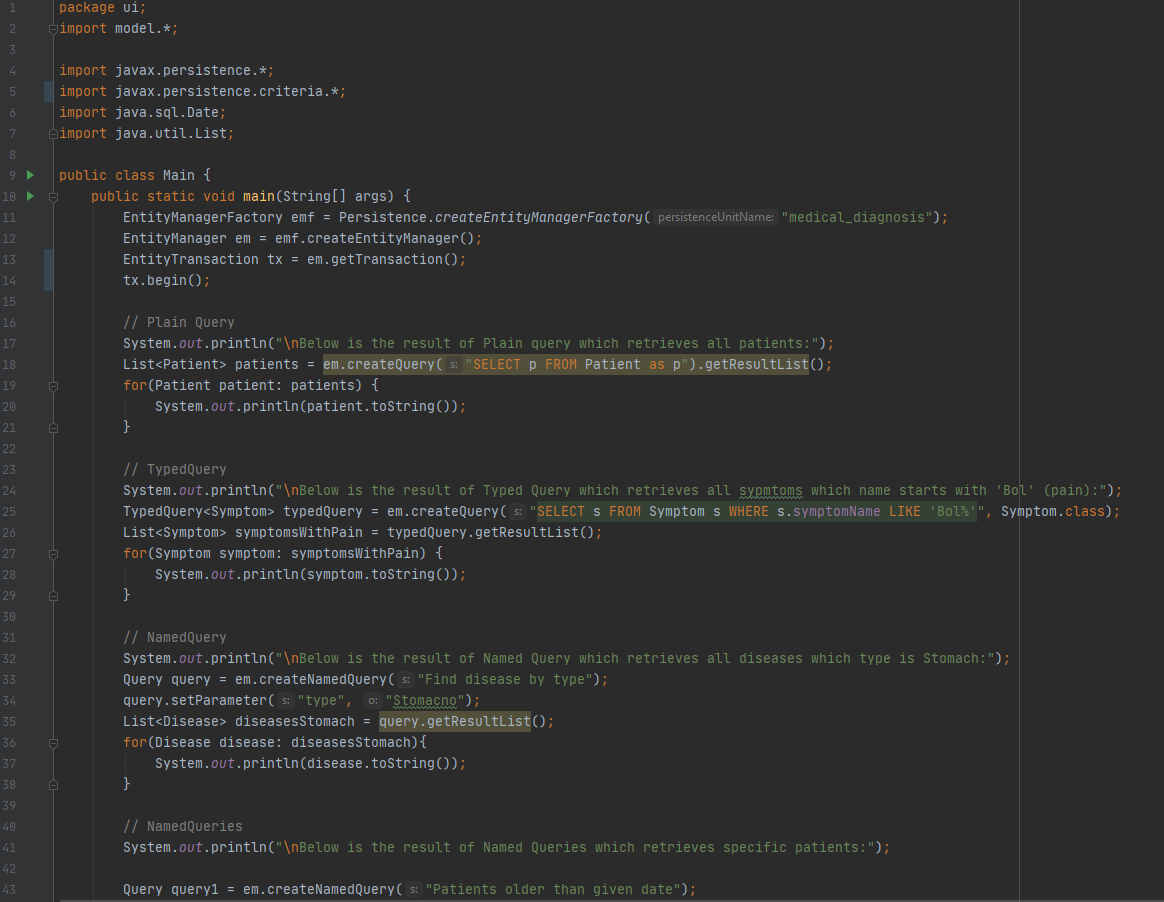
Prikaz HomeTherapy kao podklase Therapy koju i nasleđuje za vrednost Home diskriminatorske kolone.

Dodata su nova polja na sve nasleđene podatke, međutim nasleđenim podacima se mora pristupati preko gettera jer su privatnog tipa podaci.

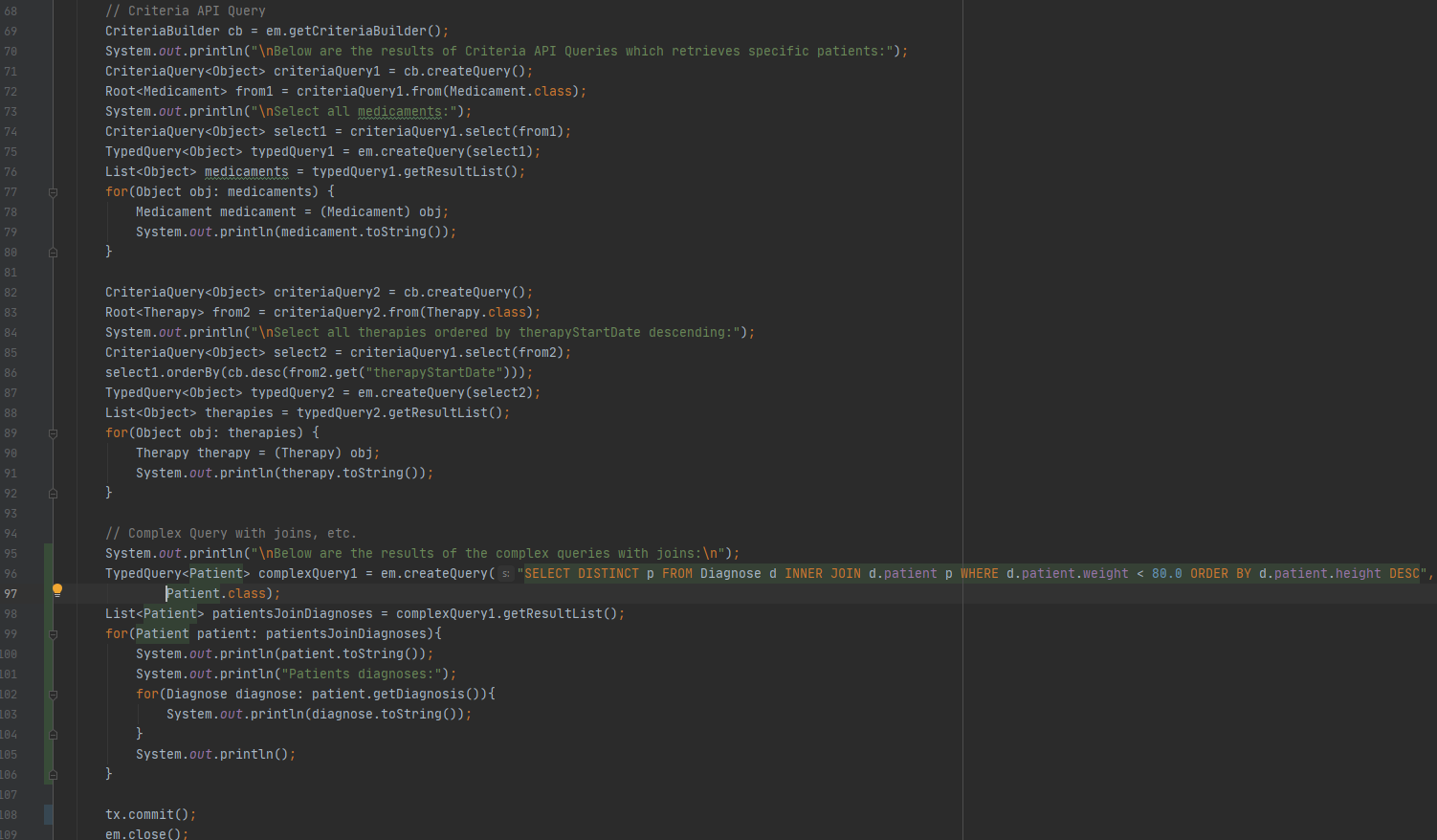


Ispod je prikaz glavne klase unutar ui foldera. Kreirali smo EntityManagerFactory preko koga smo kreirali EntityManager pomoću kog upravljamo transakcijama, pravimo upite, upravljamo radom entiteta...

Ispod toga je definicija pravljenja nekoliko vrsta upita kao što su Plain Query, TypedQuery, NamedQuery.

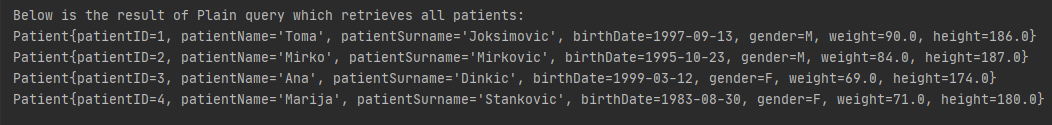


Na slici ispod vidimo prikaz joše nekoliko načina kreiranja upita. Najbolji način i najlakši za debagovanje i održavanje Criteria API Query gde imamo 2 primera. Na kraju je prikazan i jedan složeniji upit sa INNER JOIN, WHERE uslovima i ORDER BY sortiranjem za entitete Pacijenti i Dijagnoze.



1. Prikaz konzole i baze

Prikaz rezultata svih pacijenata običnog upita.



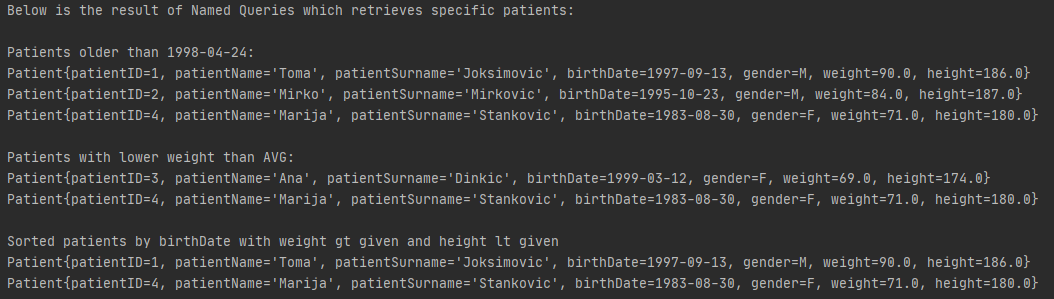
Prikaz rezultata svih simptoma koji počinju sa „Bol“ nazivom za Typed upit.



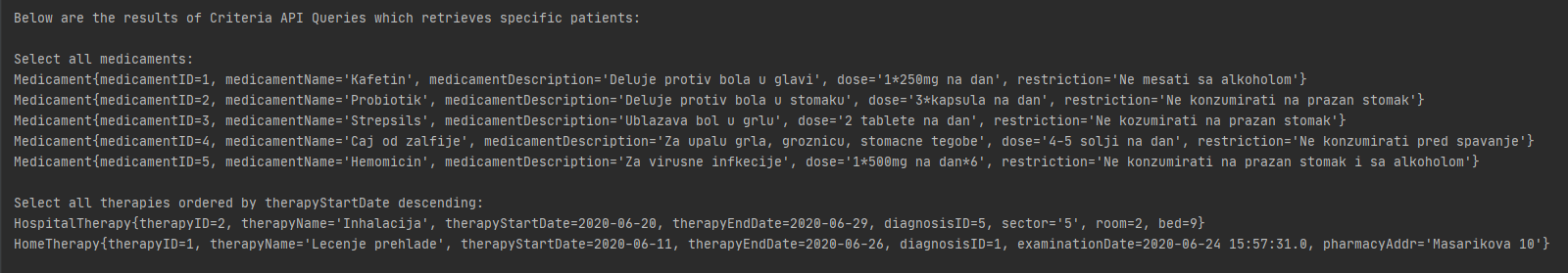
Prikaz rezultata bolesti tipa Stomačnog oboljenja korišćenjem Named upita.



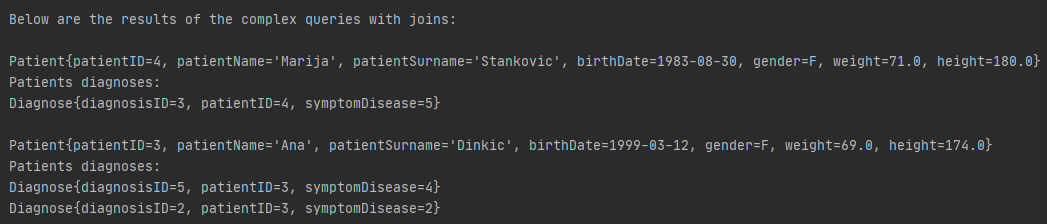
Prikaz rezultata kolekcije Named upita.



Prikaz nekoliko rezultata primenom Criteria API Query objektom CriteriaBuilder.

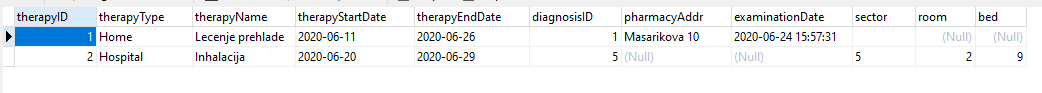


Prikaz složenog upita korišćenjem INNER JOIN metoda spajanja entiteta Patient i Diagnose.

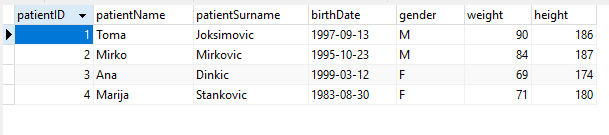


Prikaz nekoliko tabela u bazi podataka:

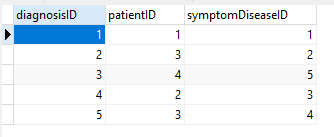
Terapije



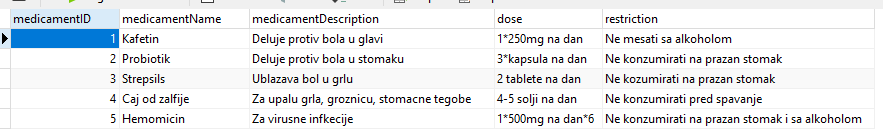
Pacijenti



Dijagnoze



Medikamenti



1. Zaključak

Ograničenja projekta su potrebno ekspertsko znanje kako bi se eventualno kreirao Ekspertski sistem ili implementirali algoritmi Mašinskog učenja na osnovu velikog broja raznovrsnih uzoraka (za koje bi bila potrebna velika količina različitih slučajeva) kako bi se donela dobra i što preciznija dijagnoza na osnovu simptoma pacijenta i odredila što efikasnija terapija.

Dodatno tehničko ograničenje je nepostajanje mogućnosti da se sistem implementira kao distribuiran koji bi radio u realnom vremenu i kao višenitna aplikacija koja komunicira sa udaljenim serverima. Za takav slučaj je potrebno bolje organizovati aplikaciju po Mikroservisnoj arhitekturi i slojevitije organizovati podatke sa interakciju sa bazom sa jasnim DAO i DTO fajlovima, kao i da kod bude lako održiv i da aplikacija može biti skalabilna.

1. Literatura

* Materijali sa predavanja/vežbi iz predmeta Praktikum – Napredno Softversko Inženjerstvo 2020
* <https://www.baeldung.com/jpa-join-types>
* <https://thorben-janssen.com/jpql/>
* <https://stackoverflow.com/>