Java Web Development

Modul 2

Termin 7

Sadržaj

1. DAL
2. API (*spring-boot-starter-jdbc*)

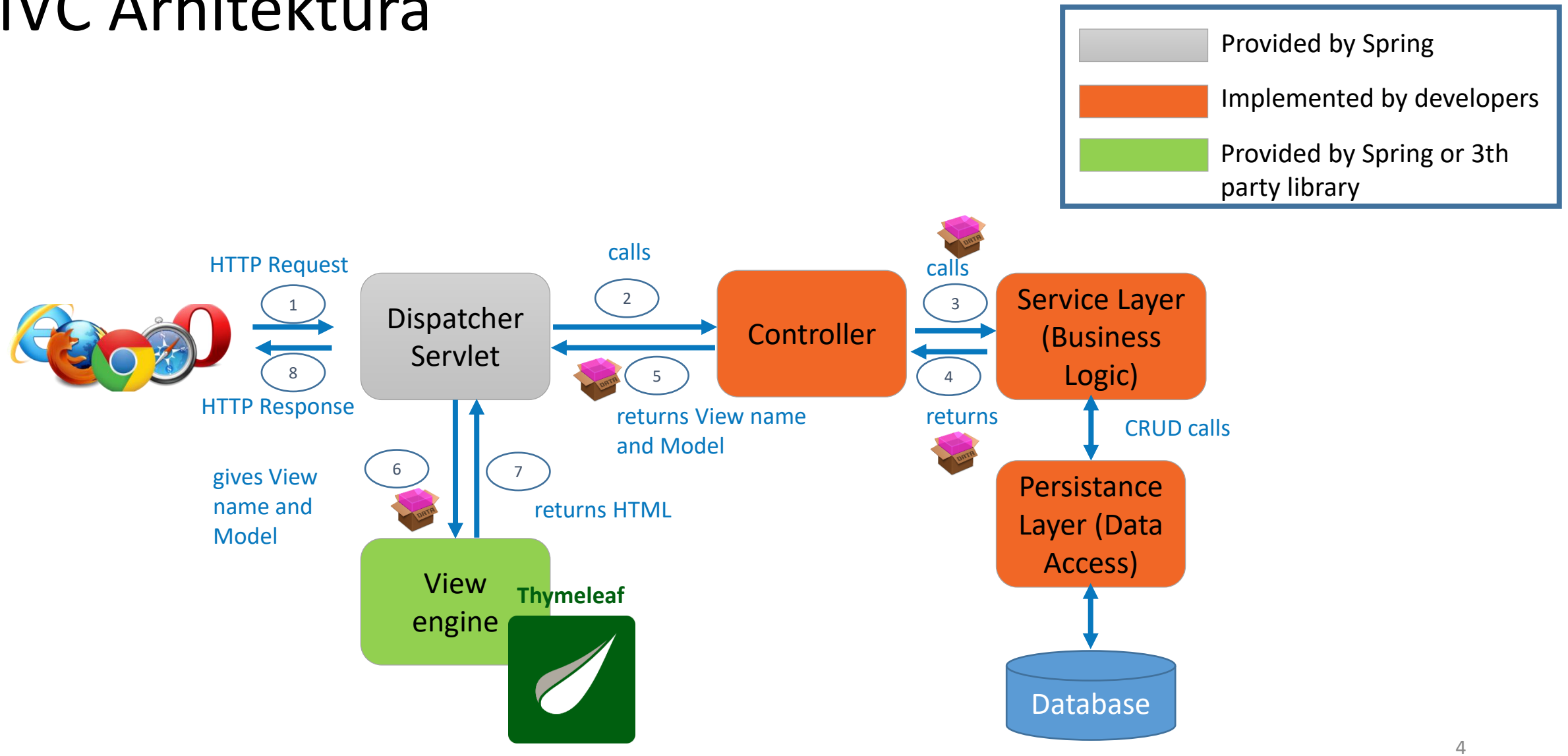
DAL

Zadatak DAL sloja je da implementira operacije za upis/čitanje u/iz izvora podataka i da ih nudi ostatku aplikacije.

Posledice:

- ostatak aplikacije postaje nezavisan od samog izvora podataka (*in-memory*, datoteke, mreža, baza podataka i sl.)
- izvor podataka postaje lako zamenljiv uz reimplementaciju samo DAL sloja, a bez izmena u ostatku aplikacije

MVC Arhitektura



DAL

DAO (*Data Access Objects*)

DAO su komponente koje sačinjavaju DAL sloj.

Tipično se za svaki tip entiteta u sistemu implementira po jedan DAO sa pripadajućim operacijama za upis/čitanje u/iz izvora podataka.

DAO tipično implementira CRUD (*Create*, *Read*, *Update*, *Delete*) operacije za pripadajući tip entiteta, ali može da implementira i složenije operacije (pretrage i sl.)

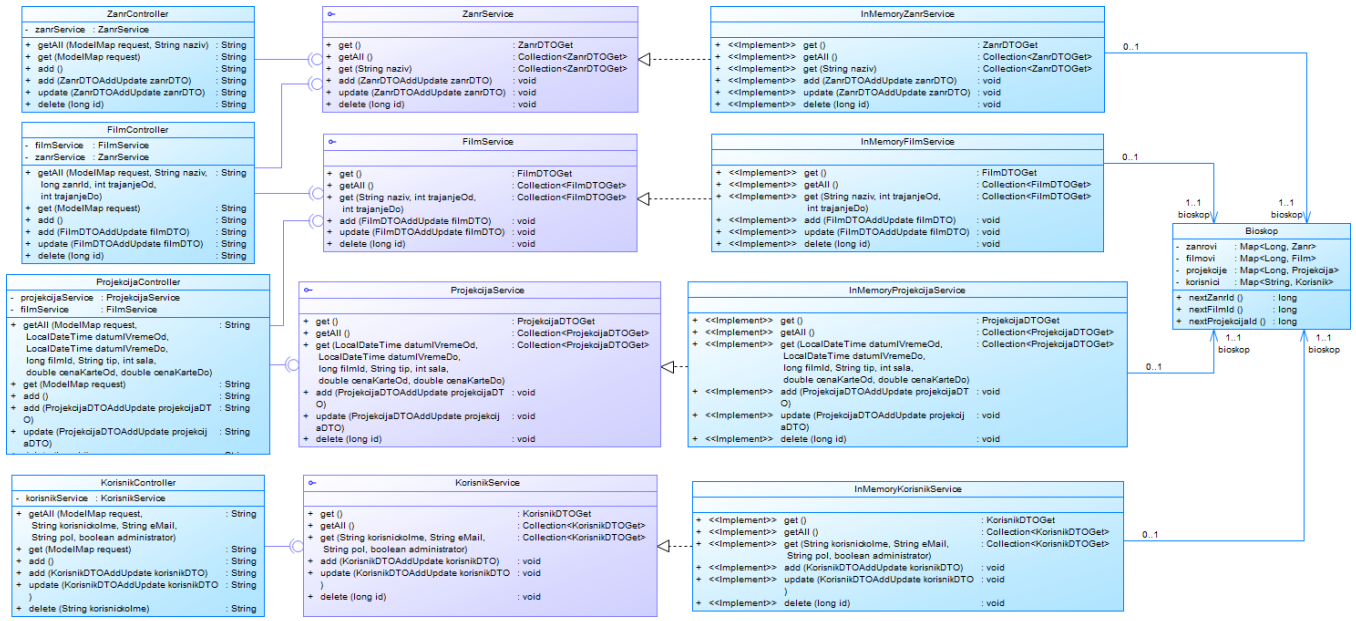
```
public interface ProjekcijaDAO {

    public Projekcija get(long id);
    public Collection<Projekcija> getAll();
    public Collection<Projekcija> get(
        LocalDateTime datumIVremeOd, LocalDateTime datumIVremeDo,
        long filmId,
        String tip,
        int sala,
        double cenaKarteOd, double cenaKarteDo);
    public void add(Projekcija projekcija);
    public void update(Projekcija projekcija);
    public void delete(long id);

}
```

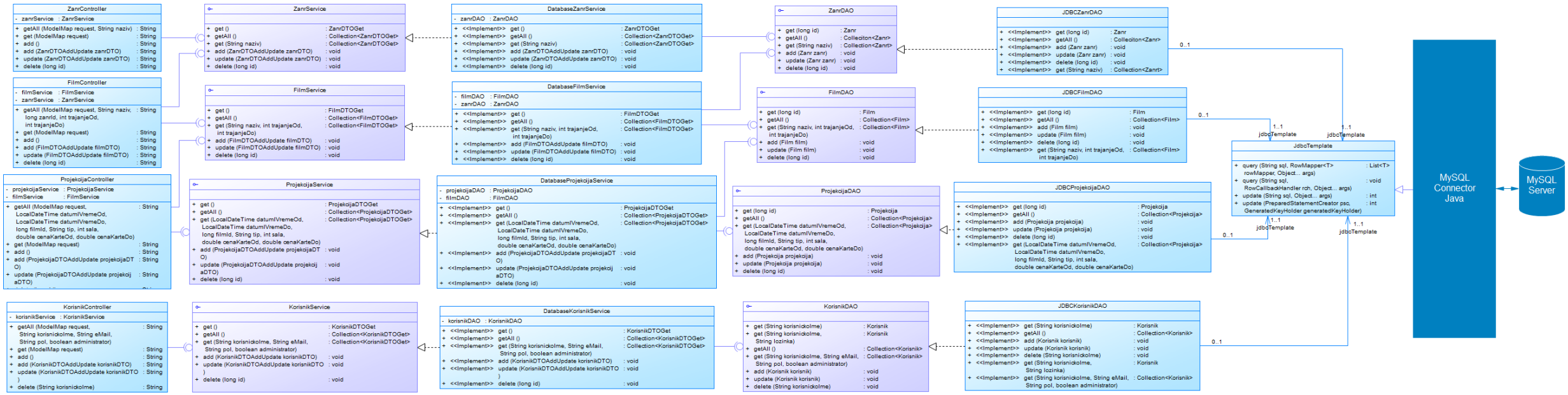
DAL

Arhitektura



DAL

Arhitektura



DAL

Arhitektura

DAO se deklarišu interfejsima.

Svaki na ovaj način deklarisan DAO može da ima jednu ili više implementacija.

Implementacije DAO-a se obeležavaju anotacijom *@Repository* koja predstavlja jednu od specijalizacija anotacije *@Component*.

Ovako anotirane implementacije DAO-a postaju vidljive *Spring* aplikaciji i dostupne *Dependency Injection* mehanizmu.

Servisi koriste DAO-e uz oslonac na interfejse.

Kao i u slučaju servisa, *@Primary* i *@Qualifier* anotacijama je moguće upravljati aktivnom implementacijom DAO-a.

interfejs

inject

```
@Controller
@RequestMapping("/projekcije")
public class ProjekcijaController {
    private final ProjekcijaService projekcijaService;
    private final FilmService filmService;

    public ProjekcijaController(
            ProjekcijaService projekcijaService,
            FilmService filmService) {
        this.projekcijaService = projekcijaService;
        this.filmService = filmService;
    }
                        ⋮
}
```

```
@Service
@Validated
public class DatabaseProjekcijaService implements ProjekcijaService
{
    private final ProjekcijaDAO projekcijaDAO;
    private final FilmDAO filmDAO;
    private final ModelMapper mapper = new ModelMapper();

    public DatabaseProjekcijaService(
            ProjekcijaDAO projekcijaDAO,
            FilmDAO filmDAO) {
        this.projekcijaDAO = projekcijaDAO;
        this.filmDAO = filmDAO;
    }
                                ⋮
}
```

```
@Repository
public class JDBCProjekcijaDAO implements ProjekcijaDAO {
    private final JdbcTemplate jdbcTemplate;
    private final FilmDAO filmDAO;

    public JDBCProjekcijaDAO(
            JdbcTemplate jdbcTemplate,
            FilmDAO filmDAO) {
        this.jdbcTemplate = jdbcTemplate;
        this.filmDAO = filmDAO;
    }
                        ⋮
}
```

DAL

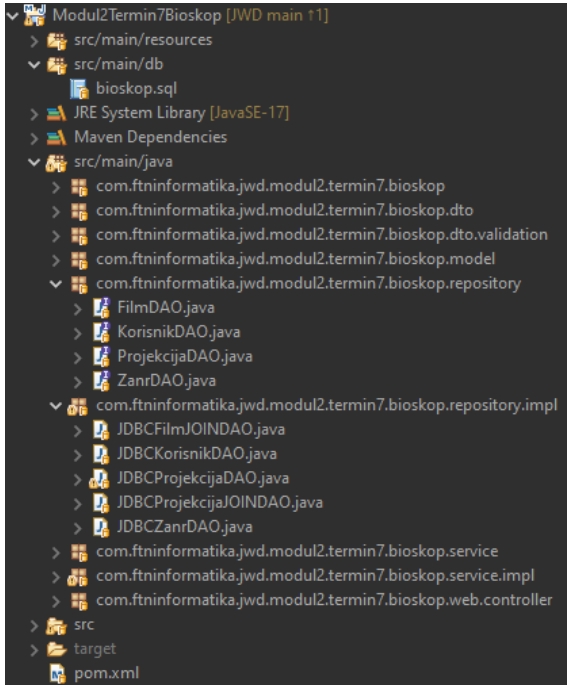
Implementacija

Nazivi DAO-a po konvenciji imaju sufiks DAO.

Deklaracije DAO-a (interfejsi) se po konvenciji navode u paketu *repository*.

Implementacije DAO-a se po konvenciji navode u paketu *repostitory.impl*.

Skripta za kreiranje/reinicijalizaciju šeme baze se po konvenciji navodi u direktorijumu *src/main/db*.



DAL

Implementacija

- *spring-boot-starter-jdbc* je API za komunikaciju sa relacionom bazom i potrebno ga je uvesti kroz *dependency* u *pom.xml*
- *mysql-connector-java* je JDBC *driver* za komunikaciju sa MySQL bazom podataka i potrebno ga je uvesti kroz *dependency* u *pom.xml*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
    <scope>runtime</scope>
</dependency>
```

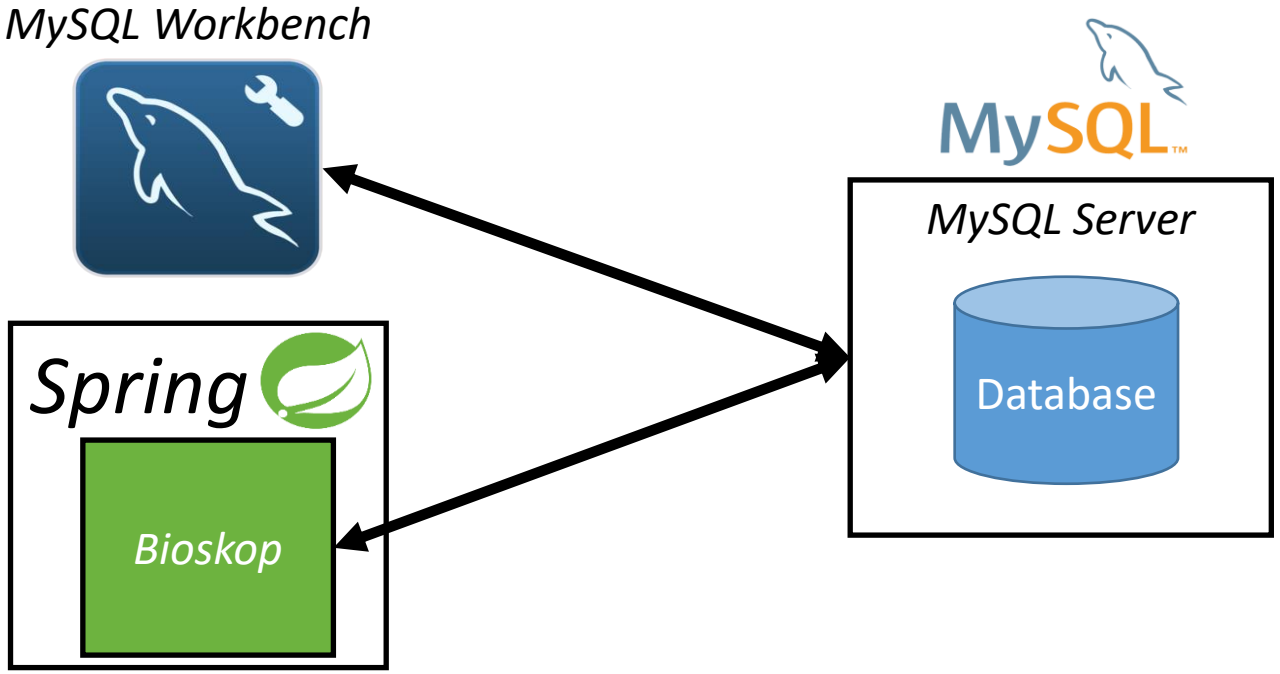
DAL

Implementacija

U *Spring* aplikaciji, *framework* je zadužen za povezivanje sa bazom podataka.

Parametri za povezivanje se navode u *application.properties* datoteci.

```
spring.datasource.url=jdbc:mysql://localhost:3306/bioskop?allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=Europe/Belgrade
spring.datasource.username=root
spring.datasource.password=root
```



(spring-boot-starter-jdbc)

JdbcTemplate

Za komunikaciju sa relacionom bazom podataka *Spring* aplikacija koristi objekat klase *JdbcTemplate* koji se kreira za vreme pokretanja aplikacije i *inject*-uje u DAO klase.

```java
@Repository
public class JDBCZanrDAO implements ZanrDAO {
    private final JdbcTemplate jdbcTemplate;

    public JDBCZanrDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
                    :
}
```

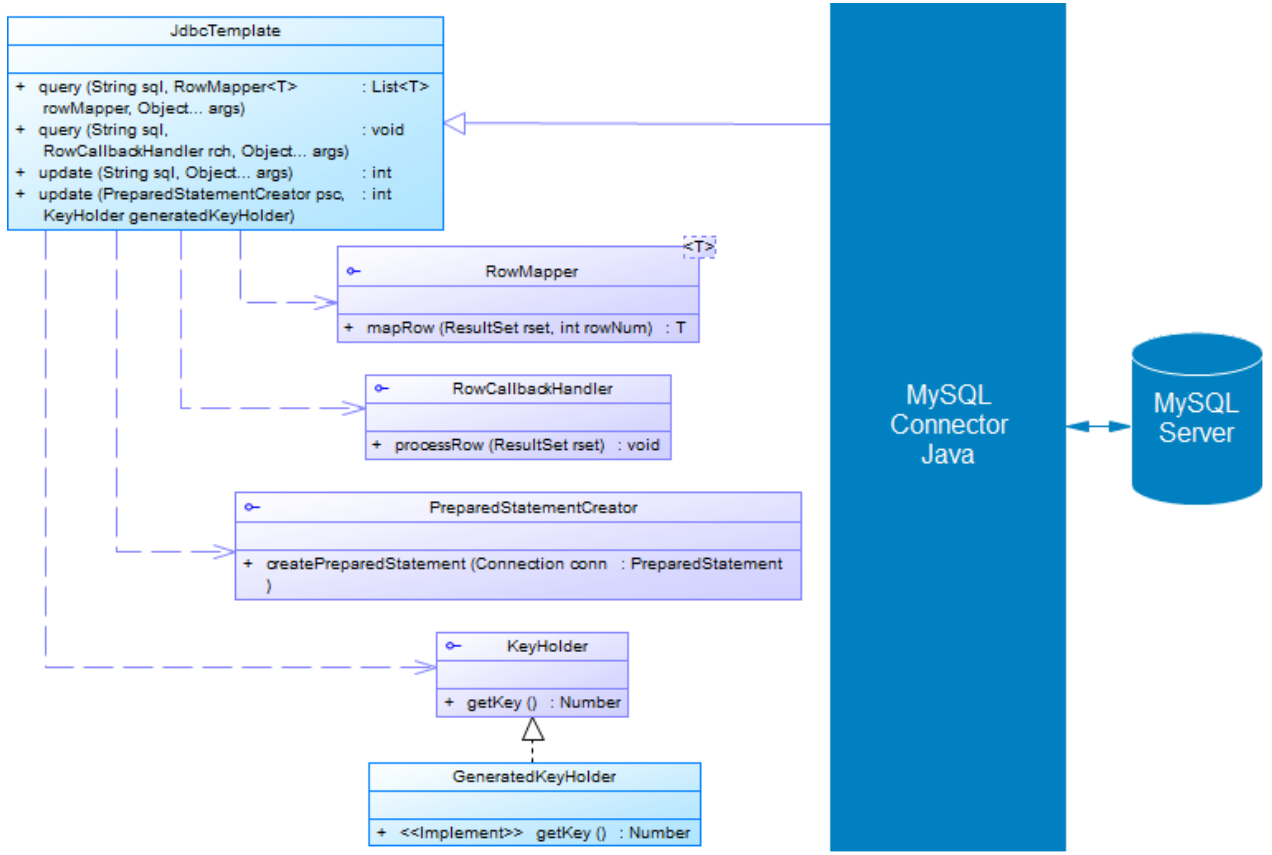
```java
@Repository
public class JDBCProjekcijaDAO implements ProjekcijaDAO {
    private final JdbcTemplate jdbcTemplate;

    public JDBCProjekcijaDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
                    :
}
```

```java
@Repository
public class JDBCFilmDAO implements FilmDAO {
    private final JdbcTemplate jdbcTemplate;

    public JDBCFilmDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
                    :
}
```

inject



(spring-boot-starter-jdbc)

JdbcTemplate (SELECT)

```
public class DatabaseProjekcijaDAO implements ProjekcijaDAO {
    private final Connection conn;
    private final FilmDAO filmDAO;
                                                                    ⋮
    @Override
    public Projekcija get(long id) throws Exception {
        Projekcija projekcija = null;

        String sql = "SELECT datumIVreme, filmId, sala, tip, cenaKarte FROM projekcije WHERE id = ?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            int param = 0;
            stmt.setLong(++param, id);
            try (ResultSet rset = stmt.executeQuery()) {
                if (rset.next()) {
                    int kolona = 0;
                    //long id = rset.getLong(++kolona);
                    LocalDateTime datumIVreme = rset.getTimestamp(++kolona).toLocalDateTime();
                    long filmId = rset.getLong(++kolona);
                    int sala = rset.getInt(++kolona);
                    String tip = rset.getString(++kolona);
                    double cenaKarte = rset.getDouble(++kolona);

                    Film film = filmDAO.get(filmId); // many-to-one veza
                    projekcija = new Projekcija(id, datumIVreme, film, sala, tip, cenaKarte);
                }
            }
        }
        return projekcija;
    }

    @Override
    public Collection<Projekcija> getAll() throws Exception {
        Collection<Projekcija> projekcije = new ArrayList<>();

        String sql = "SELECT id, datumIVreme, filmId, sala, tip, cenaKarte FROM projekcije";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            try (ResultSet rset = stmt.executeQuery()) {
                while (rset.next()) {
                    int kolona = 0;
                    long id = rset.getLong(++kolona);
                    LocalDateTime datumIVreme = rset.getTimestamp(++kolona).toLocalDateTime();
                    long filmId = rset.getLong(++kolona);
                    int sala = rset.getInt(++kolona);
                    String tip = rset.getString(++kolona);
                    double cenaKarte = rset.getDouble(++kolona);

                    Film film = filmDAO.get(filmId); // many-to-one veza
                    Projekcija projekcija = new Projekcija(id, datumIVreme, film, sala, tip, cenaKarte);
                    projekcije.add(projekcija);
                }
            }
        }
        return projekcije;
    }
                                                                    ⋮
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (SELECT)

RowMapper je generički funkcionalni interfejs čija implementacija u metodi *mapRow(ResultSet rset, int rowNum)* mapira jedan red *ResultSet*-a na objekat rezultujućeg modela i vraća ga.

Implementacija se navodi u posebnoj klasi da bi se njen objekat mogao instancirati u svim *get* metodama DAO-a.

Metoda *query(String sql, RowMapper rowMapper, Object... args)* klase *JdbcTemplate* šalje SELECT naredbu bazi podataka, a vraća listu mapiranih objekata rezultujućeg modela.

Argumenti poziva metode nakon *RowMapper*-a popunjavaju parametre SQL naredbe.

```
@Repository
public class JDBCProjekcijaDAO implements ProjekcijaDAO {
    private final JdbcTemplate jdbcTemplate;
    private final FilmDAO filmDAO;
                                                    ⋮

    private class ProjekcijaRowMapper implements RowMapper<Projekcija> {

        @Override
        public Projekcija mapRow(ResultSet rset, int rowNum) throws SQLException {
            int kolona = 0;
            long id = rset.getLong(++kolona);
            LocalDateTime datumIVreme = rset.getTimestamp(++kolona).toLocalDateTime();
            long filmId = rset.getLong(++kolona);
            String tip = rset.getString(++kolona);
            int sala = rset.getInt(++kolona);
            double cenaKarte = rset.getDouble(++kolona);

           Film film = filmDAO.get(filmId); // many-to-one veza
           Projekcija projekcija = new Projekcija(id, datumIVreme, film, tip, sala, cenaKarte);
           return projekcija;
        }

    }

    @Override
    public Projekcija get(long id) {
        String sql =
            "SELECT id, datumIVreme, filmId, tip, sala, cenaKarte FROM projekcije " +
            "WHERE id = ?";

        List<Projekcija> rezultat = jdbcTemplate.query(sql, new ProjekcijaRowMapper(), id);
        return (!rezultat.isEmpty())? rezultat.get(0): null;
    }

    @Override
    public Collection<Projekcija> getAll() {
        String sql = "SELECT id, datumIVreme, filmId, tip, sala, cenaKarte FROM projekcije";
        return jdbcTemplate.query(sql, new ProjekcijaRowMapper());
    }
                                                    ⋮
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (SELECT)

```
public class DatabaseFilmDAO implements FilmDAO {
    private final Connection conn;
                                                ⋮
    @Override
    public Collection<Film> getAll() throws Exception {
        Map<Long, Film> filmovi = new LinkedHashMap<>();

        String sql =
            "SELECT f.id, f.naziv, f.trajanje, z.id, z.naziv FROM filmovi f " +
            "LEFT JOIN filmZanr fz ON f.id = fz.filmId " +
            "LEFT JOIN zanrovi z ON fz.zanrId = z.id";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            try (ResultSet rset = stmt.executeQuery()) {
                while (rset.next()) {
                int kolona = 0;
                long fId = rset.getLong(++kolona);
                String fNaziv = rset.getString(++kolona);
                int fTrajanje = rset.getInt(++kolona);

                Film film = filmovi.get(fId);
                if (film == null) {
                    filmovi.put(film.getId(), film);
                }
                // many-to-many veza
                long zId = rset.getLong(++kolona);
                if (zId != 0) {
                    String zNaziv = rset.getString(++kolona);
                    Zanr zanr = new Zanr(zId, zNaziv);
                    film.addZanr(zanr);
                }
            }
        }
        return filmovi.values();
    }
                                                ⋮
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (SELECT)

RowCallbackHandler je funkcionalni interfejs čija implementacija u metodi *processRow(ResultSet rset)* mapira jedan red *ResultSet*-a na povezane objekte rezultujućeg modela i ne vraća ništa.

Implementacija je zadužena za čuvanje rezultata u privatnoj kolekciji, što omogućuje punu kontrolu nad kreiranjem rezultujće kolekcije i izbegavanje kreiranja duplikata.

Implementacija se navodi u posebnoj klasi da bi se njen objekat mogao instancirati u svim *get* metodama DAO-a.

Metoda *query(String sql, RowCallbackHandler rch, Object... args)* klase *JdbcTemplate* šalje SELECT naredbu bazi podataka, a ne vraća ništa.

Argumenti poziva metode nakon *RowCallbackHandler*-a popunjavaju parametre SQL naredbe.

```
@Repository
public class JDBCFilmDAO implements FilmDAO {
    private final JdbcTemplate jdbcTemplate;
                                                    ⋮

    private static class FilmRowCallbackHandler implements RowCallbackHandler {
        private final Map<Long, Film> filmovi = new LinkedHashMap<>();

        @Override
        public void processRow(ResultSet rset) throws SQLException {
            int kolona = 0;
            long fId = rset.getLong(++kolona);
            String fNaziv = rset.getString(++kolona);
            int fTrajanje = rset.getInt(++kolona);
            Film film = filmovi.get(fId);
            if (film == null) {
                film = new Film(fId, fNaziv, fTrajanje);
                filmovi.put(fId, film);
            }
            // many-to-many veza
            long zId = rset.getLong(++kolona);
            if (zId != 0) {
                String zNaziv = rset.getString(++kolona);

                Zanr zanr = new Zanr(zId, zNaziv);
                film.addZanr(zanr);
            }
        }

        public List<Film> getFilmovi() { return new ArrayList<>(filmovi.values()); }
    }


    @Override
    public Collection<Film> getAll() {
        FilmRowCallbackHandler rowCallbackHandler = new FilmRowCallbackHandler();
        String sql =
            "SELECT f.id, f.naziv, f.trajanje, z.id, z.naziv FROM filmovi f " +
            "LEFT JOIN filmZanr fz ON f.id = fz.filmId " +
            "LEFT JOIN zanrovi z ON z.id = fz.zanrId " +
            "ORDER BY f.id";
        jdbcTemplate.query(sql, rowCallbackHandler);

        return rowCallbackHandler.getFilmovi();
    }
                                                    ⋮
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (SELECT)

RowCallbackHandler je funkcionalni interfejs čija implementacija u metodi *processRow(ResultSet rset)* mapira jedan red *ResultSet*-a na povezane objekte rezultujućeg modela i ne vraća ništa.

Implementacija je zadužena za čuvanje rezultata u privatnoj kolekciji, što omogućuje punu kontrolu nad kreiranjem rezultujće kolekcije i izbegavanje kreiranja duplikata.

Implementacija se navodi u posebnoj klasi da bi se njen objekat mogao instancirati u svim *get* metodama DAO-a.

Metoda *query(String sql, RowCallbackHandler rch, Object... args)* klase *JdbcTemplate* šalje SELECT naredbu bazi podataka, a ne vraća ništa.

Argumenti poziva metode nakon *RowCallbackHandler*-a popunjavaju parametre SQL naredbe.

Rezultat se čita iz objekta *RowCallbackHandler*-a.

```
@Repository
public class JDBCFilmDAO implements FilmDAO {
    private final JdbcTemplate jdbcTemplate;
                                                    ⋮

    private static class FilmRowCallbackHandler implements RowCallbackHandler {
        private final Map<Long, Film> filmovi = new LinkedHashMap<>();

        @Override
        public void processRow(ResultSet rset) throws SQLException {
            int kolona = 0;
            long fId = rset.getLong(++kolona);
            String fNaziv = rset.getString(++kolona);
            int fTrajanje = rset.getInt(++kolona);
            Film film = filmovi.get(fId);
            if (film == null) {
                film = new Film(fId, fNaziv, fTrajanje);
                filmovi.put(fId, film);
            }
            // many-to-many veza
            long zId = rset.getLong(++kolona);
            if (zId != 0) {
                String zNaziv = rset.getString(++kolona);

                Zanr zanr = new Zanr(zId, zNaziv);
                film.addZanr(zanr);
            }
        }

        public List<Film> getFilmovi() { return new ArrayList<>(filmovi.values()); }
    }


    @Override
    public Collection<Film> getAll() {
        FilmRowCallbackHandler rowCallbackHandler = new FilmRowCallbackHandler();
        String sql =
            "SELECT f.id, f.naziv, f.trajanje, z.id, z.naziv FROM filmovi f " +
            "LEFT JOIN filmZanr fz ON f.id = fz.filmId " +
            "LEFT JOIN zanrovi z ON z.id = fz.zanrId " +
            "ORDER BY f.id";
        jdbcTemplate.query(sql, rowCallbackHandler);

        return rowCallbackHandler.getFilmovi();
    }
                                                    ⋮
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (INSERT, UPDATE, DELETE)

```java
public class DatabaseProjekcijaDAO implements ProjekcijaDAO {
    private final Connection conn;
    private final FilmDAO filmDAO;
                                                    ⋮
    @Override
    public void add(Projekcija projekcija) throws Exception {
        String sql = "INSERT INTO projekcije (datumIVreme, filmId, sala, tip, cenaKarte) VALUES (?, ?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            int param = 0;
            stmt.setTimestamp(++param, Timestamp.valueOf(projekcija.getDatumIVreme()));
            stmt.setLong(++param, projekcija.getFilm().getId()); // many-to-one veza
            stmt.setInt(++param, projekcija.getSala());
            stmt.setString(++param, projekcija.getTip());
            stmt.setDouble(++param, projekcija.getCenaKarte());

            stmt.executeUpdate();
        }
    }

    @Override
    public void update(Projekcija projekcija) throws Exception {
        String sql =
            "UPDATE projekcije SET datumIVreme = ?, filmId = ?, sala = ?, tip = ?, cenaKarte = ? " +
            "WHERE id = ?";

        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            int param = 0;
            stmt.setTimestamp(++param, Timestamp.valueOf(projekcija.getDatumIVreme()));
            stmt.setLong(++param, projekcija.getFilm().getId()); // many-to-one veza
            stmt.setInt(++param, projekcija.getSala());
            stmt.setString(++param, projekcija.getTip());
            stmt.setDouble(++param, projekcija.getCenaKarte());
            stmt.setLong(++param, projekcija.getId());

            stmt.executeUpdate();
        }
    }

    @Override
    public void delete(long id) throws Exception {
        String sql = "DELETE FROM projekcije WHERE id = ?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            int param = 0;
            stmt.setLong(++param, id);

            stmt.executeUpdate();
        }
    }
                                                    ⋮
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (INSERT, UPDATE, DELETE)

Metoda *update(String sql, Object... args)* klase *JdbcTemplate* šalje INSERT, UPDATE ili DELETE naredbu bazi podataka, a vraća broj izmenjenih redova u bazi.

Argumenti poziva metode nakon SQL-a popunjavaju parametre SQL naredbe.

```
@Repository
public class JDBCProjekcijaDAO implements ProjekcijaDAO {
    private final JdbcTemplate jdbcTemplate;
    private final FilmDAO filmDAO;
                                                    :

    @Override
    public void add(Projekcija projekcija) {
        String sql =
            "INSERT INTO projekcije (datumIVreme, filmId, tip, sala, cenaKarte) " +
            "VALUES (?, ?, ?, ?, ?)";
        jdbcTemplate.update(sql,
            projekcija.getDatumIVreme(),
            projekcija.getFilm().getId(), // many-to-one veza
            projekcija.getTip(),
            projekcija.getSala(),
            projekcija.getCenaKarte());
    }

    @Override
    public void update(Projekcija projekcija) {
        String sql =
            "UPDATE projekcije "  +
            "SET datumIVreme = ?  filmId = ?, tip = ?, sala = ?, cenaKarte = ? "  +
            "WHERE id = ?";
        jdbcTemplate.update(sql,
            projekcija.getDatumIVreme(),
            projekcija.getFilm().getId(), // many-to-one veza
            projekcija.getTip(),
            projekcija.getSala(),
            projekcija.getCenaKarte(),
            projekcija.getId());
    }

    @Override
    public void delete(long id) {
        String sql = "DELETE FROM projekcije WHERE id = ?";
        jdbcTemplate.update(sql, id);
    }
                                                    :
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (INSERT, UPDATE, DELETE)

```
public class DatabaseFilmDAO implements FilmDAO {
    private final Connection conn;
    private final FilmZanrDAO filmZanrDAO;
                                                    :
    @Override
    public void add(Film film) throws Exception {
        // transakcija
        try {
            conn.setAutoCommit(false);
            conn.commit();

            // NAREDBA 1 (jednom): dodavanje u tabelu filmovi
            String sql = "INSERT INTO filmovi (naziv, trajanje) VALUES (?, ?)";
            try (PreparedStatement stmt = conn.prepareStatement(
                    sql,
                    PreparedStatement.RETURN_GENERATED_KEYS)) {
                int param = 0;
                stmt.setString(++param, film.getNaziv());
                stmt.setInt(++param, film.getTrajanje());

                boolean uspeh = stmt.executeUpdate() == 1;
                if (uspeh) {
                    // many-to-many veza
                    try (ResultSet rset = stmt.getGeneratedKeys()) {
                        if (rset.next()) {
                            long id = rset.getLong(1);
                            for (Zanr itZanr: film.getZanrovi()) {
                                // NAREDBA 2 (više puta): dodavanje u tabelu filmZanr
                                filmZanrDAO.add(id, itZanr.getId());
                            }
                        }
                    }
                }
            }
            conn.commit();
        } catch (Exception ex) {
            conn.rollback();
            throw ex;
        } finally {
            conn.setAutoCommit(true);
        }
    }
                                                    :
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (INSERT, UPDATE, DELETE)

PreparedStatementCreator je funkcionalni interfejs čija implementacija u metodi *createPreparedStatement(Connection conn)* kreira naredbu tipa *PreparedStatement* i vraća je, što omogućuje punu kontrolu nad kreiranjem naredbe.

Ukoliko je potreban pristup parametrima metode, implementacija se navodi kao anonimna klasa unutar te metode.

Metoda *update(PreparedStatementCreator pch, KeyHolder keyHolder)* klase *JdbcTemplate* šalje INSERT, UPDATE ili DELETE naredbu bazi podataka, a vraća broj izmenjenih redova u bazi.

```
@Repository
public class JDBCFilmDAO implements FilmDAO {
    private final JdbcTemplate jdbcTemplate;

                                                    :

    @Override
    @Transactional
    public void add(Film film) {
        PreparedStatementCreator preparedStatementCreator = new PreparedStatementCreator() {

            @Override
            public PreparedStatement createPreparedStatement(Connection conn)
                    throws SQLException {
                String sql = "INSERT INTO filmovi (naziv, trajanje) VALUES (?, ?)";

                PreparedStatement preparedStatement = conn.prepareStatement(
                        sql,
                        Statement.RETURN_GENERATED_KEYS);
                int param = 0;
                preparedStatement.setString(++param, film.getNaziv());
                preparedStatement.setInt(++param, film.getTrajanje());

                return preparedStatement;
            }
        };
        GeneratedKeyHolder keyHolder = new GeneratedKeyHolder();

        boolean uspeh = jdbcTemplate.update(preparedStatementCreator, keyHolder) == 1;
        if (uspeh) {
            String sql = "INSERT INTO filmZanr (filmId, zanrId) VALUES (?, ?)";
            for (Zanr itZanr: film.getZanrovi()) {
                jdbcTemplate.update(sql, keyHolder.getKey(), itZanr.getId());
            }
        }
    }

                                                    :
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (INSERT, UPDATE, DELETE)

PreparedStatementCreator je funkcionalni interfejs čija implementacija u metodi *createPreparedStatement(Connection conn)* kreira naredbu tipa *PreparedStatement* i vraća je, što omogućuje punu kontrolu nad kreiranjem naredbe.

Ukoliko je potreban pristup parametrima metode, implementacija se navodi kao anonimna klasa unutar te metode.

Metoda *update(PreparedStatementCreator pch, KeyHolder keyHolder)* klase *JdbcTemplate* šalje INSERT, UPDATE ili DELETE naredbu bazi podataka, a vraća broj izmenjenih redova u bazi.

Nakon izvršene naredbe, generisani ključ će biti vraćen i upisan u objekat klase *GeneratedKeyHolder* odakle ga je moguće pročitati i upotrebiti u ostalim naredbama.

```
@Repository
public class JDBCFilmDAO implements FilmDAO {
    private final JdbcTemplate jdbcTemplate;

                                                    :

    @Override
    @Transactional
    public void add(Film film) {
        PreparedStatementCreator preparedStatementCreator = new PreparedStatementCreator() {

            @Override
            public PreparedStatement createPreparedStatement(Connection conn)
                    throws SQLException {
                String sql = "INSERT INTO filmovi (naziv, trajanje) VALUES (?, ?)";

                PreparedStatement preparedStatement = conn.prepareStatement(
                        sql,
                        Statement.RETURN_GENERATED_KEYS);
                int param = 0;
                preparedStatement.setString(++param, film.getNaziv());
                preparedStatement.setInt(++param, film.getTrajanje());

                return preparedStatement;
            }
        };
        GeneratedKeyHolder keyHolder = new GeneratedKeyHolder();

        boolean uspeh = jdbcTemplate.update(preparedStatementCreator, keyHolder) == 1;
        if (uspeh) {
            String sql = "INSERT INTO filmZanr (filmId, zanrId) VALUES (?, ?)";
            for (Zanr itZanr: film.getZanrovi()) {
                jdbcTemplate.update(sql, keyHolder.getKey(), itZanr.getId());
            }
        }
    }

                                                    :
}
```

(spring-boot-starter-jdbc)

JdbcTemplate (INSERT, UPDATE, DELETE)

PreparedStatementCreator je funkcionalni interfejs čija implementacija u metodi *createPreparedStatement(Connection conn)* kreira naredbu tipa *PreparedStatement* i vraća je, što omogućuje punu kontrolu nad kreiranjem naredbe.

Ukoliko je potreban pristup parametrima metode, implementacija se navodi kao anonimna klasa unutar te metode.

Metoda *update(PreparedStatementCreator pch, KeyHolder keyHolder)* klase *JdbcTemplate* šalje INSERT, UPDATE ili DELETE naredbu bazi podataka, a vraća broj izmenjenih redova u bazi.

Nakon izvršene naredbe, generisani ključ će biti vraćen i upisan u objekat klase *GeneratedKeyHolder* odakle ga je moguće pročitati i upotrebiti u ostalim naredbama.

Anotacija *@Transactional* grupiše sve naredbe u metodi transakciju.

```
@Repository
public class JDBCFilmDAO implements FilmDAO {
    private final JdbcTemplate jdbcTemplate;

                                                    :

    @Override
    @Transactional
    public void add(Film film) {
        PreparedStatementCreator preparedStatementCreator = new PreparedStatementCreator() {

            @Override
            public PreparedStatement createPreparedStatement(Connection conn)
                    throws SQLException {
                String sql = "INSERT INTO filmovi (naziv, trajanje) VALUES (?, ?)";

                PreparedStatement preparedStatement = conn.prepareStatement(
                        sql,
                        Statement.RETURN_GENERATED_KEYS);
                int param = 0;
                preparedStatement.setString(++param, film.getNaziv());
                preparedStatement.setInt(++param, film.getTrajanje());

                return preparedStatement;
            }
        };
        GeneratedKeyHolder keyHolder = new GeneratedKeyHolder();

        boolean uspeh = jdbcTemplate.update(preparedStatementCreator, keyHolder) == 1;
        if (uspeh) {
            String sql = "INSERT INTO filmZanr (filmId, zanrId) VALUES (?, ?)";
            for (Zanr itZanr: film.getZanrovi()) {
                jdbcTemplate.update(sql, keyHolder.getKey(), itZanr.getId());
            }
        }
    }
                                                    :
}
```

Primer

- *com.ftninformatika.jwd.modul2.termin6.bioskop*

Zadatak 1

Po ugledu na primer *com.ftninformatika.jwd.modul2.termin7.bioskop* implementirati DAL u zadatku *com.ftninformatika.jwd.modul2.termin7.dostava:*

1. Uvesti *sping-boot-starter-jdbc dependency* u *pom.xml* datoteku
2. Uvesti *mysql-connector-java dependency* u *pom.xml* datoteku
3. Definisati DAO interfejse u paketu *repository*
4. Započeti DAO implementacije u paketu *repository.impl*
5. Započeti implementacije database servisa u paketu *service.impl* i obeležiti ih *@Primary* anotacijom
6. *Inject*-ovati klasu *JdbcTemplate* u započete DAO implementacije
7. *Inject*-ovati započete DAO implementacije u započete implementacije *database* servisa uz oslonac na njihove interfejse
8. Implementirati jednu po jednu metodu jednog po jednog DAO-a, a zatim servisa i testirati funkcionalnost

Dodatni materijali

- https://spring.io/guides/gs/relational-data-access/
- https://mkyong.com/spring-boot/spring-boot-jdbc-examples/