



ftninformatika

Java Web Development

Modul 2

Termin 6

Sadržaj

1. Servisni sloj
2. DTO
3. Validacija
4. Pretrage

Servisni sloj

Poslovna logika

U poslovnu logiku spada rukovanje podacima poput: validacije, izračunavanja, povezivanja, i sl.

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {

    :

    @GetMapping("") // bez @ResponseBody
    public String getAll(ModelMap request,
        @RequestParam(required = false, defaultValue = "0") long zanrId) {
        Collection<Film> rezultat = new ArrayList<>();
        for (Film itFilm: bioskop.getFilmovi().values()) { // pretraga filmova po žanr id
            for (Zanr itZanr: itFilm.getZanrovi()) {
                if (zanrId == 0 || itZanr.getId() == zanrId) {
                    rezultat.add(itFilm);
                    break; // prekid samo unutrašnje petlje
                }
            }
        }
        request.addAttribute("filmovi", rezultat);
        return "filmovi"; // forwarding na template
    }

    :
}
```

Servisni sloj

Poslovna logika

U poslovnu logiku spada rukovanje podacima poput: validacije, izračunavanja, povezivanja, i sl.

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {

    :

    @PostMapping("/dodavanje")
    public String add(
        @RequestParam String naziv,
        @RequestParam int trajanje,
        @RequestParam long[] zanrIds) {

        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: zanrIds) { // pretraga žanrova po id
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) { // da li je bar jedan žanr pronađen?
            return "redirect:/filmovi"; // spreči dodavanje
        }

        long id = bioskop.nextFilmId();
        Film film = new Film(id, naziv, trajanje);
        film.setZanrovi(zanrovi); // povezivanje
        bioskop.getFilmovi().put(id, film);

        return "redirect:/filmovi";
    }

    :
}
```

Servisni sloj

Poslovna logika

U poslovnu logiku spada rukovanje podacima poput: validacije, izračunavanja, povezivanja, i sl.

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {

    :

    @PostMapping("/brisanje")
    public String delete(@RequestParam long id) {
        // kaskadno brisanje
        Iterator<Entry<Long, Projekcija>> itEntryProjekcija = bioskop.getProjekcije().entrySet().iterator();
        while (itEntryProjekcija.hasNext()) {
            Projekcija itProjekcija = itEntryProjekcija.next().getValue();
            if (itProjekcija.getFilm().getId() == id) {
                itEntryProjekcija.remove();
            }
        }
        bioskop.getFilmovi().remove(id);

        return "redirect:/filmovi";
    }

    :
}
```

Servisni sloj

Poslovna logika

U poslovnu logiku spada rukovanje podacima poput: validacije, izračunavanja, povezivanja, i sl.

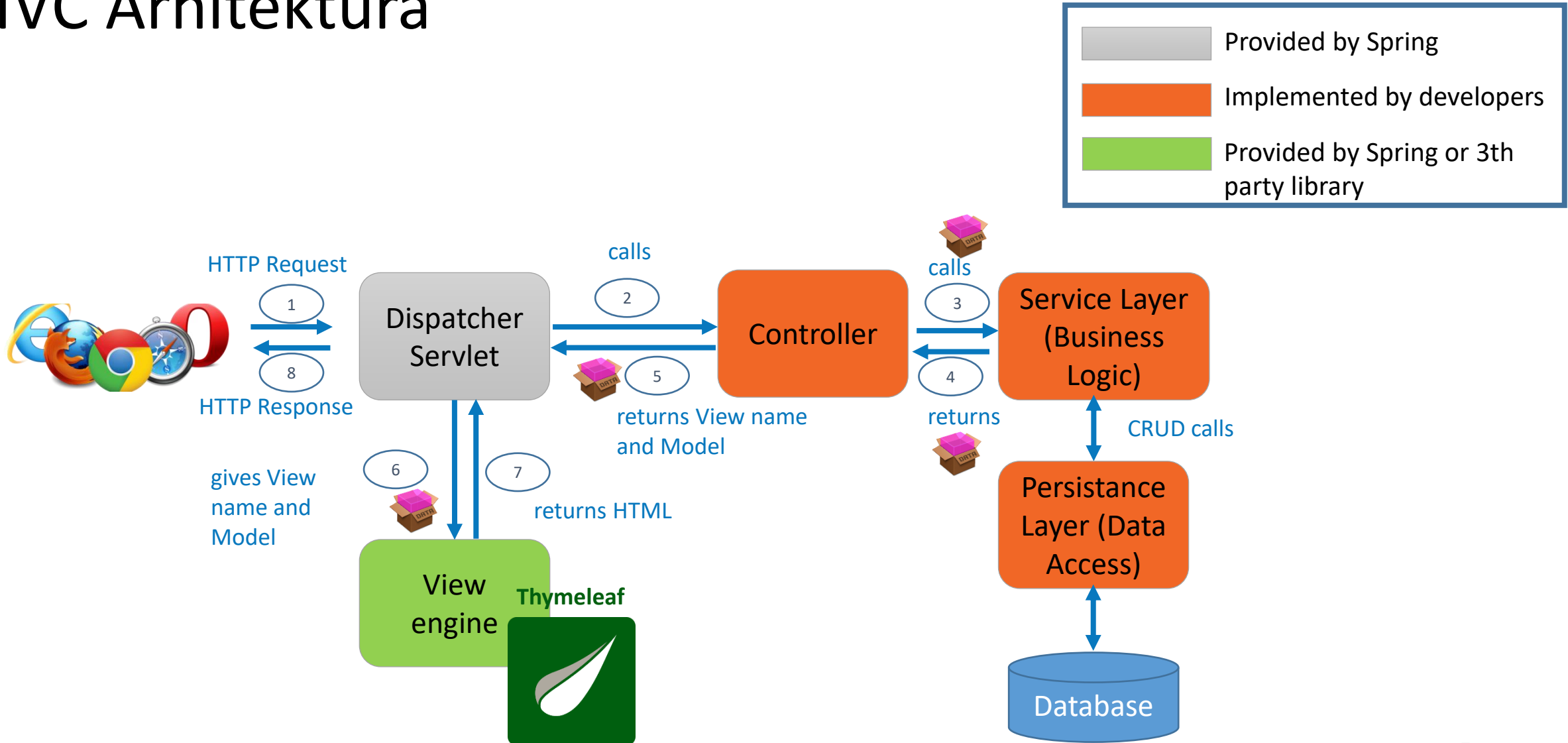
Ove operacije bi trebalo da su implementirane nezavisno od prezentacionog sloja aplikacije (*controller-a*), ali i DAL sloja.

Na taj način se poslovna logika odvaja od načina prezentacije podataka, ali i načina skladištenja podataka.

Servisni sloj služi da implementira poslovnu logiku.

Servisni sloj

MVC Arhitektura



Servisni sloj

Servisi

Servisi su komponente koje sačinjavaju servisni sloj.

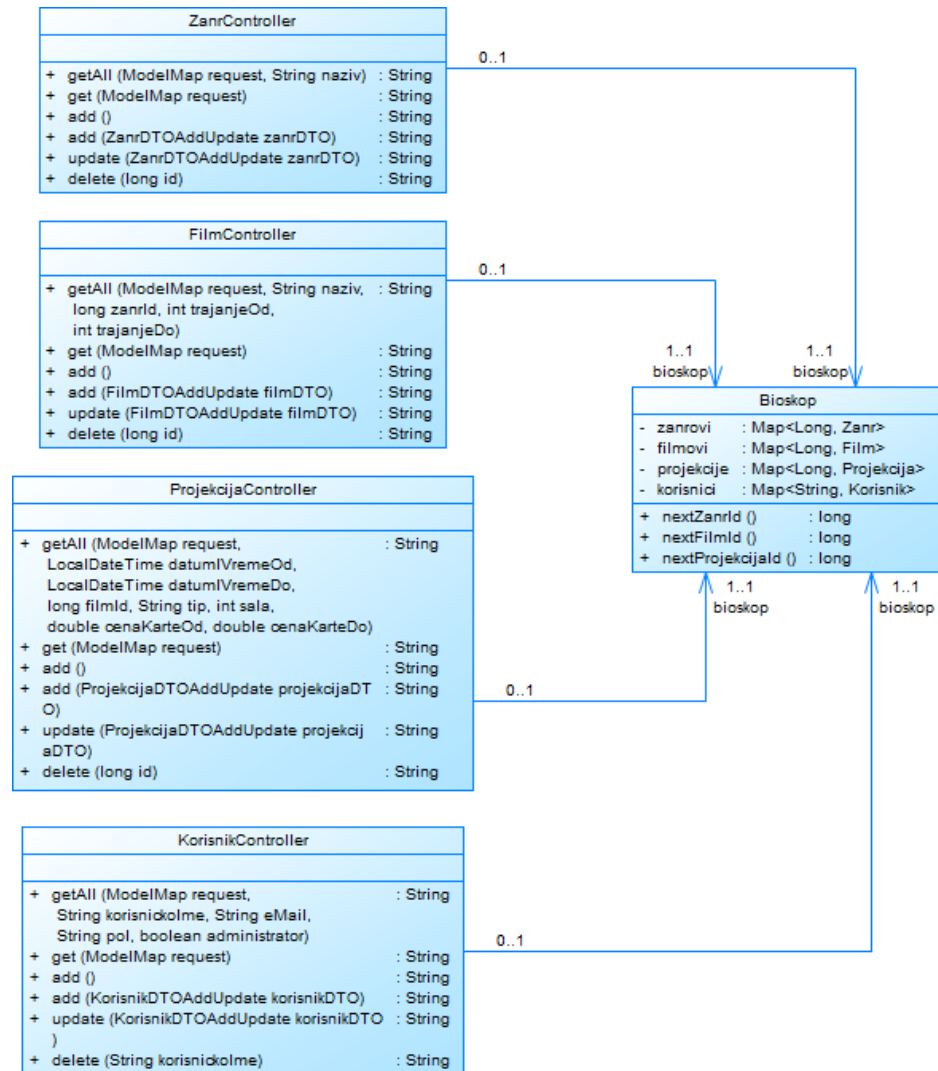
Tipično se za svaki tip entiteta u sistemu implementira po jedan servis sa pripadajućom poslovnom logikom.

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

```
public interface FilmService {  
  
    public Film get(long id);  
    public Collection<Film> getAll();  
    public Collection<Film> get(String naziv, long zanrId, int trajanjeOd, int trajanjeDo);  
    public void add(Film film);  
    public void update(Film film);  
    public void delete(long id);  
  
}
```

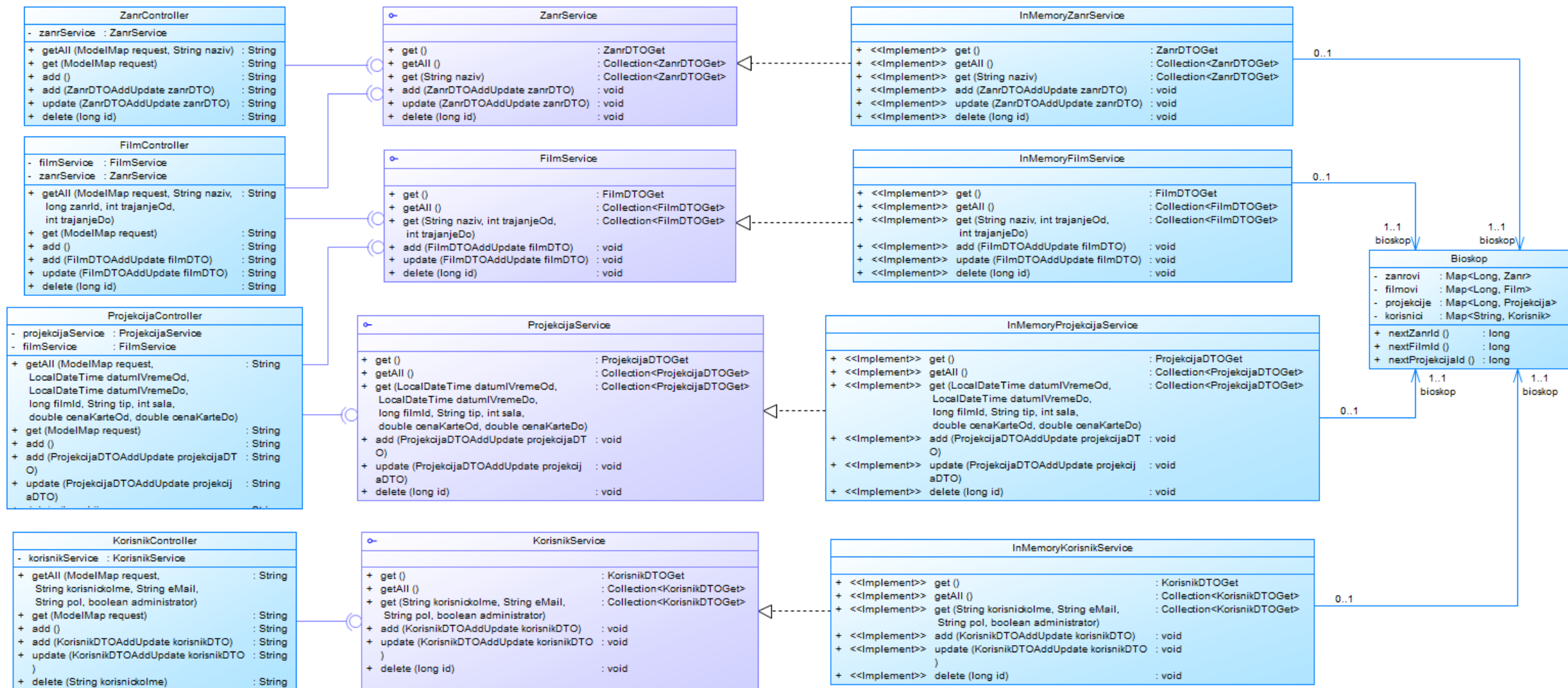

Servisni sloj

Arhitektura



Servisni sloj

Arhitektura



Servisni sloj

Arhitektura

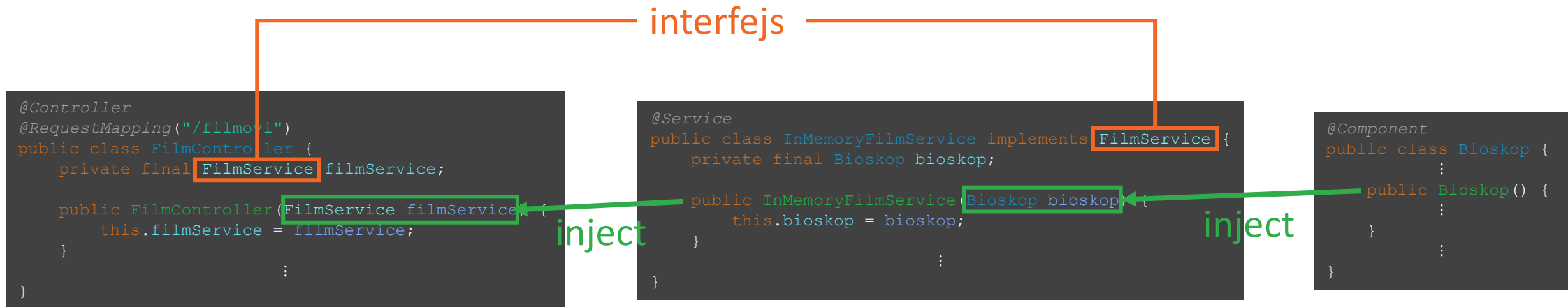
Servisi se deklariraju interfejsima.

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

Implementacije servisa se obeležavaju anotacijom *@Service* koja predstavlja jednu od specijalizacija anotacije *@Component*.

Ovako anotirane implementacije servisa postaju vidljive *Spring* aplikaciji i dostupne *Dependency Injection* mehanizmu.

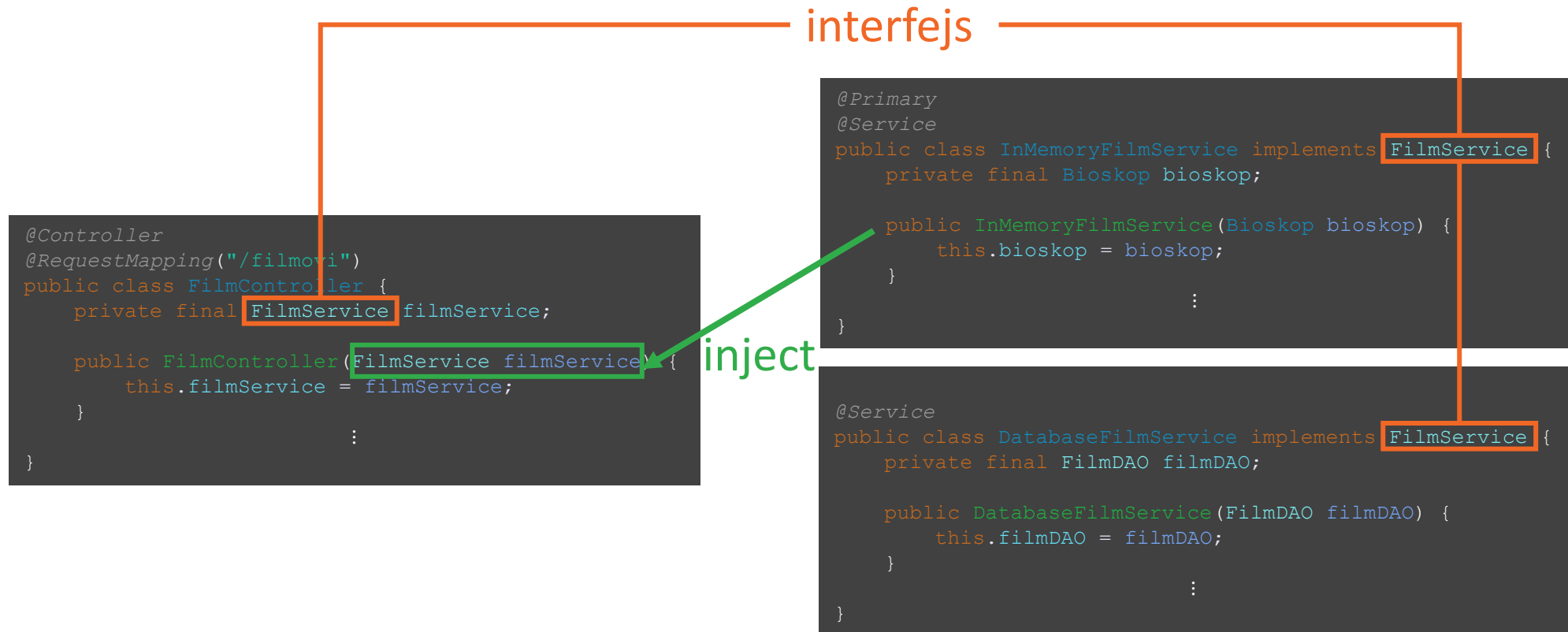
Controller-i koriste servise uz oslonac na interfejse.



Servisni sloj

Arhitektura

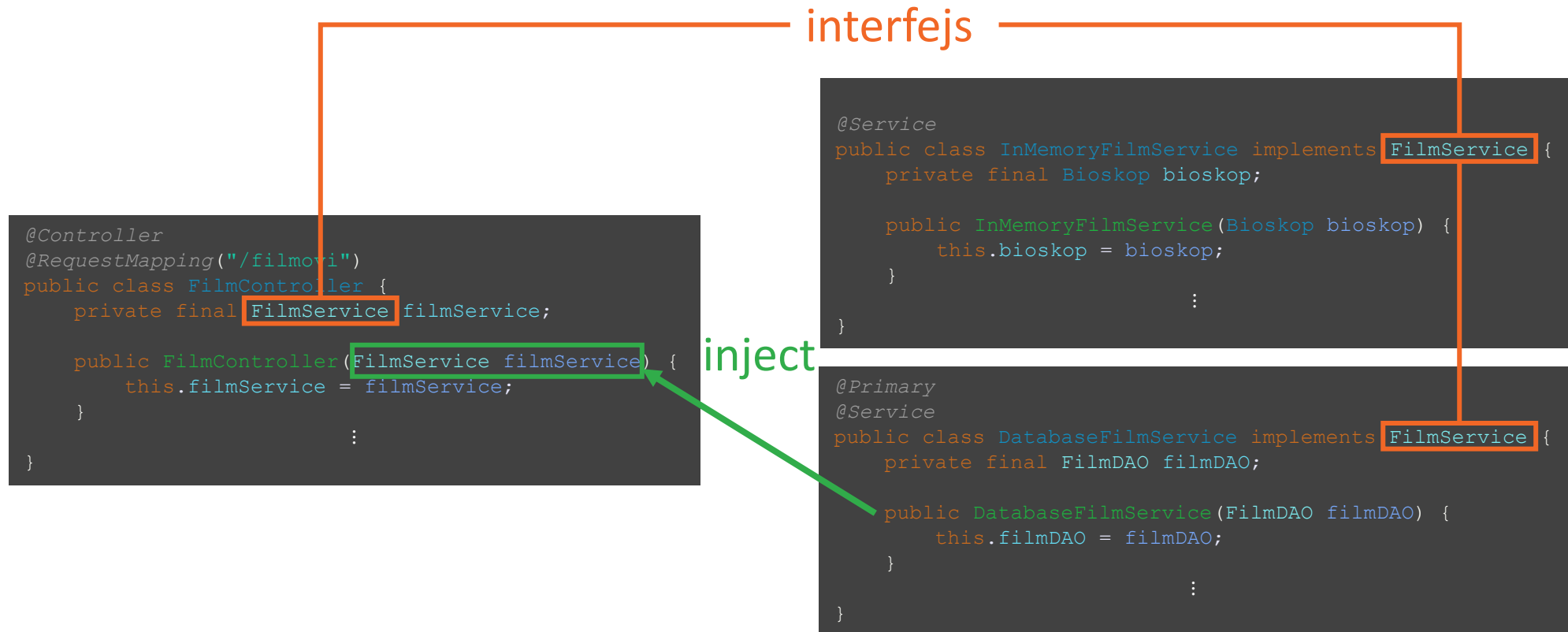
@Primary anotacijom je moguće birati aktivnu implementaciju servisa.



Servisni sloj

Arhitektura

@Primary anotacijom je moguće birati aktivnu implementaciju servisa.



Servisni sloj

Arhitektura

@Qualifier anotacijom je moguće birati aktivnu implementaciju servisa.

interfejs

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;

    public FilmController(@Qualifier("inMemory") FilmService filmService) {
        this.filmService = filmService;
    }
}
```

inject

```
@Qualifier("inMemory")
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;

    public InMemoryFilmService(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
}
```

```
@Qualifier("database")
@Service
public class DatabaseFilmService implements FilmService {
    private final FilmDAO filmDAO;

    public DatabaseFilmService(FilmDAO filmDAO) {
        this.filmDAO = filmDAO;
    }
}
```

Servisni sloj

Arhitektura

@Qualifier anotacijom je moguće birati aktivnu implementaciju servisa.

interfejs

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;

    public FilmController(@Qualifier("database") FilmService filmService) {
        this.filmService = filmService;
    }
}
```

inject

```
@Qualifier("inMemory")
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;

    public InMemoryFilmService(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
}
```

```
@Qualifier("database")
@Service
public class DatabaseFilmService implements FilmService {
    private final FilmDAO filmDAO;

    public DatabaseFilmService(FilmDAO filmDAO) {
        this.filmDAO = filmDAO;
    }
}
```

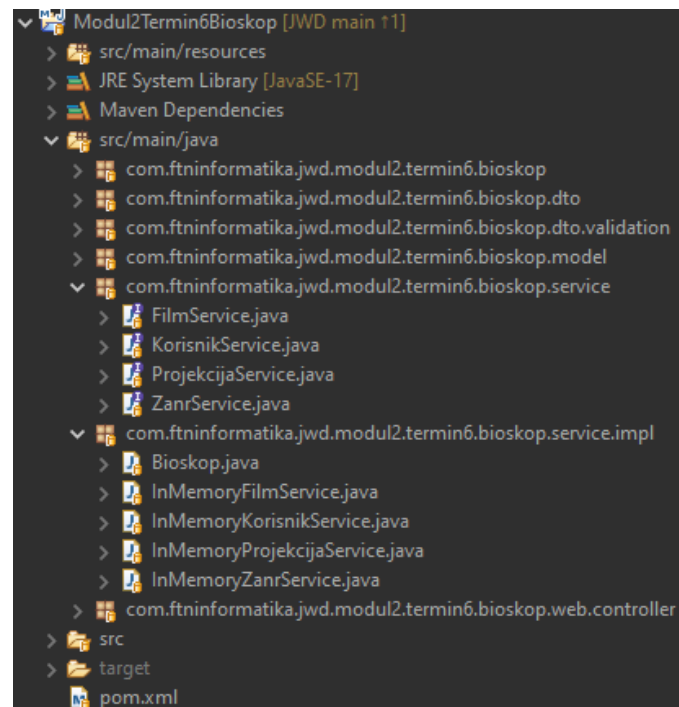
Servisni sloj

Implementacija

Nazivi servisa po konvenciji imaju sufiks *Service*.

Deklaracije servisa (interfejsi) se po konvenciji navode u paketu *service*.

Implementacije servisa se po konvenciji navode u paketu *service.impl*.



DTO

Problem

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final Bioskop bioskop;
    :

    @PostMapping("/dodavanje")
    public String add(
        @RequestParam String naziv,
        @RequestParam int trajanje,
        @RequestParam long[] zanrIds) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: zanrIds) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            return "redirect:/filmovi";
        }
        long id = bioskop.nextFilmId();
        Film film = new Film(id, naziv, trajanje);
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);

        return "redirect:/filmovi";
    }
    :
}
```

potrebno je
implementirati poslovnu logiku
u metodi servisa

DTO

Problem

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;

    :

    @PostMapping("/dodavanje")
    public String add(
        @RequestParam String naziv,
        @RequestParam int trajanje,
        @RequestParam long[] zanrIds) {
        Film film = new Film(0, naziv, trajanje, zanrIds);
        filmService.add(film);

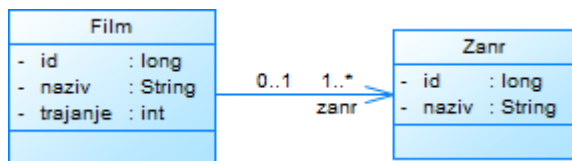
        return "redirect:/filmovi";
    }
}
```

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;

    :

    @Override
    public void add(Film film) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: zanrIds) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronađen nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        film.setId(id);
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    :
}
```



postoji nepodudaranje između ulaznih/prezentovanih podataka i modela

DTO

Rešenje

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;

    :

    @PostMapping("/dodavanje")
    public String add(
        @RequestParam String naziv,
        @RequestParam int trajanje,
        @RequestParam long[] zanrIds) {
        FilmDTO film = new FilmDTO(0, naziv, trajanje, zanrIds);
        filmService.add(film);

        return "redirect:/filmovi";
    }

    :
}
```

FilmDTO	
- id	: long
- naziv	: String
- trajanje	: int
- zanrIds	: long[]

```
public class FilmDTO {
    private long id;
    private String naziv;
    private int trajanje;
    private long[] zanrIds;

    :

}
```

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;

    :

    @Override
    public void add(FilmDTO filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronađen nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        Film film = new Film(id, filmDTO.getNaziv(), filmDTO.getTrajanje());
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    :
}
```

definisati namenski model za potrebe prenosa podataka između slojeva aplikacije

DTO

Pogodnost

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;

    :

    @PostMapping("/dodavanje")
    public String add(@ModelAttribute FilmDTO filmDTO) {
        filmService.add(filmDTO);
        return "redirect:/filmovi";
    }

    :
}
```

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;

    :

    @Override
    public void add(FilmDTO filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronađen nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        Film film = new Film(id, filmDTO.getNaziv(), filmDTO.getTrajanje());
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    :
}
```

naziv=Novi&zanrIds=3&zanrIds=5&trajanje=100

```
public class FilmDTO {
    private long id;
    private String naziv;
    private int trajanje;
    private long[] zanrIds;

    :
}
```

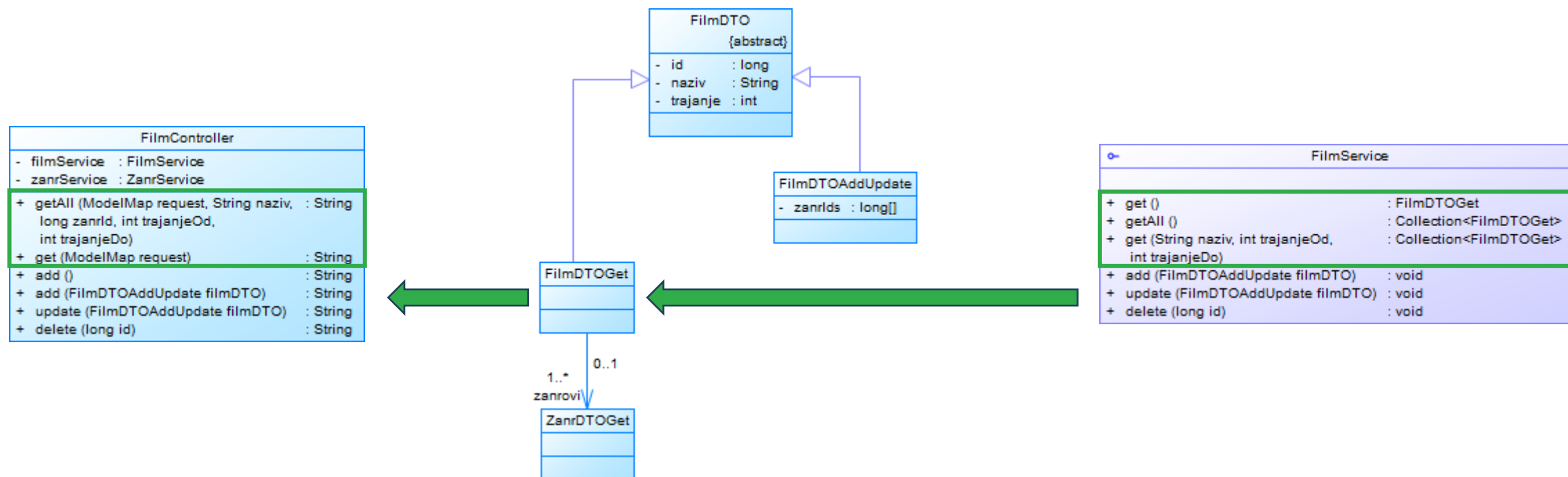
@ModelAttribute anotacija od aplikacije zahteva automatsko mapiranje (kopiranje) parametara zahteva u attribute anotiranog parametra metode *controller*-a (po nazivu atributa).

DTO

DTO (*Data Transfer Objects*) su objekti namenskih klasa kojima se podaci prenose između slojeva aplikacije.

Moguće je definisati posebne specijalizacije ovih klasa za komunikaciju u oba smera između slojeva aplikacije, odnosno za pojedinačne metode uslužnog sloja.

Pored prilagođavanja oblika podataka među slojevima, DTO služe i sprečavanju prenošenja viška podataka što između ostalog poboljšava i bezbednost aplikacije, naročito u REST arhitekturi.

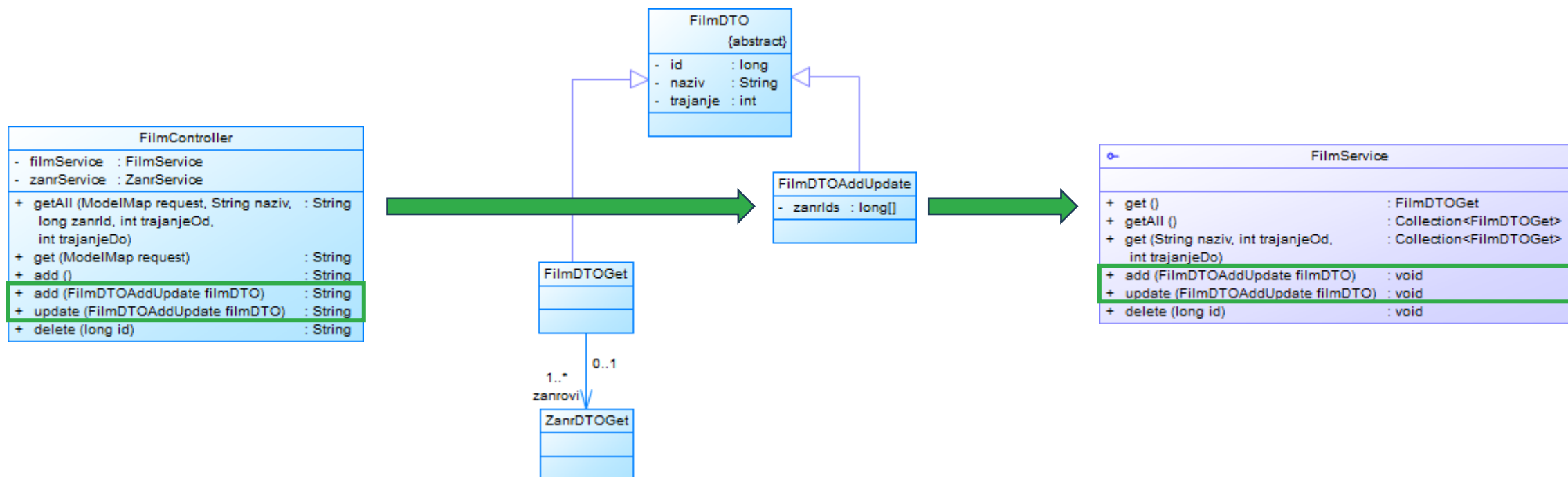


DTO

DTO (*Data Transfer Objects*) su objekti namenskih klasa kojima se podaci prenose između slojeva aplikacije.

Moguće je definisati posebne specijalizacije ovih klasa za komunikaciju u oba smera između slojeva aplikacije, odnosno za pojedinačne metode uslužnog sloja.

Pored prilagođavanja oblika podataka među slojevima, DTO služe i sprečavanju prenošenja viška podataka što između ostalog poboljšava i bezbednost aplikacije, naročito u REST arhitekturi.

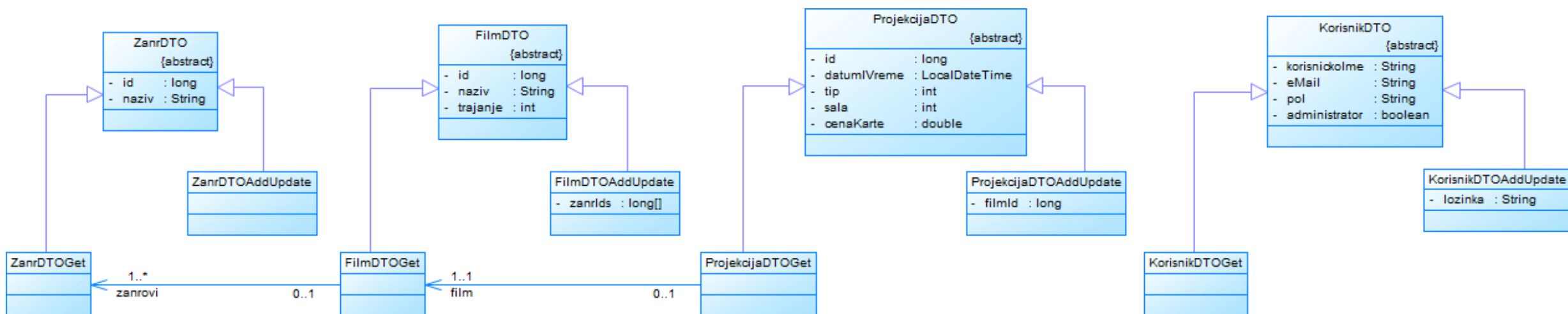


DTO

DTO (*Data Transfer Objects*) su objekti namenskih klasa kojima se podaci prenose između slojeva aplikacije.

Moguće je definisati posebne specijalizacije ovih klasa za komunikaciju u oba smera između slojeva aplikacije, odnosno za pojedinačne metode uslužnog sloja.

Pored prilagođavanja oblika podataka među slojevima, DTO služe i sprečavanju prenošenja viška podataka što između ostalog poboljšava i bezbednost aplikacije, naročito u REST arhitekturi.

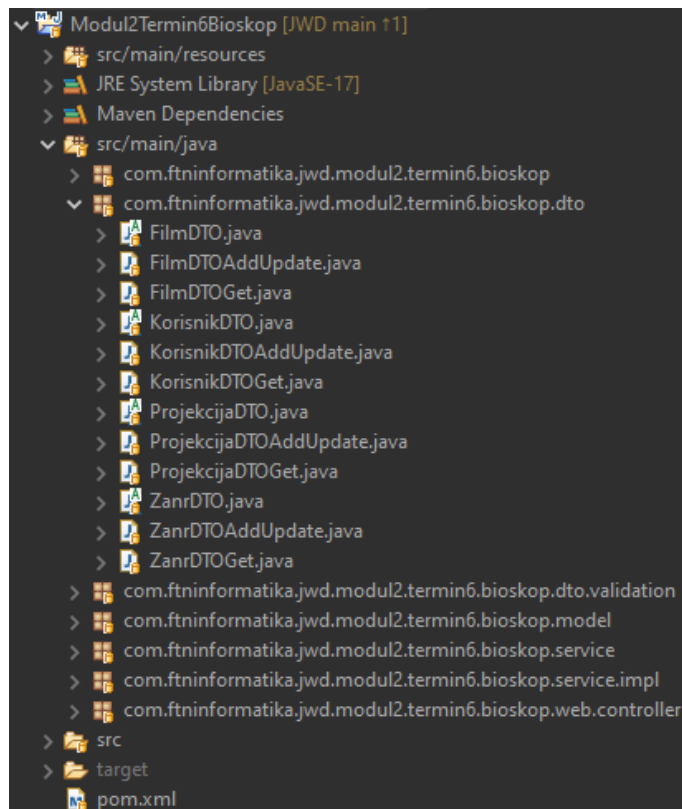


DTO

Implementacija

DTO klase po konvenciji u svom nazivu sadrže skraćenicu “DTO”.

DTO klase se po konvenciji navode u paketu *dto*.



DTO

ModelMapper

ModelMapper je klasa iz istoimene biblioteke: <https://modelmapper.org/>

Omogućava automatsko mapiranje (kopiranje) atributa između objekata različitih klasa (u osnovnoj primeni po nazivu atributa).

Klase čiji se objekti mapiraju na ovaj način moraju imati **konstruktor bez parametara** i **getter-e** i **setter-e** za sve attribute koje je potrebno mapirati.

Biblioteku je potrebno uvesti kroz *dependency* u datoteci *pom.xml*.

```
<dependency>
  <groupId>org.modelmapper</groupId>
  <artifactId>modelmapper</artifactId>
  <version>3.1.1</version>
</dependency>
```

DTO

ModelMapper (Add)

<Type> map(Object source, Class<Type> destinationType)

- ova verzija *map* metode kopira attribute objekta neke klase (*source*) u novi objekat druge odabrane klase (*destinationType*) i vraća ga kao povratnu vrednost

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;

    :

    @Override
    public void add(FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronađen nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        Film film = new Film(id, filmDTO.getNaziv(), filmDTO.getTrajanje());
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    :
}
```

DTO

ModelMapper (Add)

<Type> map(Object source, Class<Type> destinationType)

- ova verzija *map* metode kopira attribute objekta neke klase (*source*) u novi objekat druge odabrane klase (*destinationType*) i vraća ga kao povratnu vrednost

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;
    private final ModelMapper mapper = new ModelMapper();

    :

    @Override
    public void add(FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronađen nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        Film film = mapper.map(filmDTO, Film.class);
        film.setId(id);
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    :
}
```

DTO

ModelMapper (Update)

void map(Object source, Object destination)

- ova verzija *map* metode kopira attribute objekta neke (*source*) klase u postojeći objekat druge klase (*destination*).

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;

    :

    @Override
    public void update(FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronađen nijedan žanr!");
        }
        long id = filmDTO.getId();
        Film film = bioskop.getFilmovi().get(id);
        film.setNaziv(filmDTO.getNaziv());
        film.setTrajanje(filmDTO.getTrajanje());
        film.setZanrovi(zanrovi);
    }

    :
}
```

DTO

ModelMapper (Update)

void map(Object source, Object destination)

- ova verzija *map* metode kopira attribute objekta neke (*source*) klase u postojeći objekat druge klase (*destination*).

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;
    private final ModelMapper mapper = new ModelMapper();

    :

    @Override
    public void update(FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronađen nijedan žanr!");
        }
        long id = filmDTO.getId();
        Film film = bioskop.getFilmovi().get(id);
        mapper.map(filmDTO, film);
        film.setZanrovi(zanrovi);
    }

    :
}
```

DTO

ModelMapper (Get)

Mapiranje povratnih vrednosti metoda servisa iz objekata modela u DTO je poželjno izdvojiti u posebne metode servisa jer se upotrebljavaju na više mesta.

Type createDTO(Type source)

Collection<Type> createDTO(Collection<Model> source)

Potrebna je jedna ovakva metoda za mapiranje objekta modela na DTO i jedna za mapiranje kolekcije objekata modela na kolekciju DTO.

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;
    private final ModelMapper mapper = new ModelMapper();

    :

    private FilmDTOGet createDTO(Film film) {
        return mapper.map(film, FilmDTOGet.class);
    }

    private Collection<FilmDTOGet> createDTO(Collection<Film> filmovi) {
        Collection<FilmDTOGet> filmDTOS = new ArrayList<>();
        for (Film itFilm: filmovi) {
            FilmDTOGet filmDTO = createDTO(itFilm);
            filmDTOS.add(filmDTO);
        }
        return filmDTOS;
    }

    @Override
    public FilmDTOGet get(long id) {
        Film film = bioskop.getFilmovi().get(id);
        if (film == null) {
            throw new NoSuchElementException("Film nije pronaden!");
        }
        return createDTO(film);
    }

    @Override
    public Collection<FilmDTOGet> getAll() {
        Collection<Film> filmovi = bioskop.getFilmovi().values();
        return createDTO(filmovi);
    }

    :
}
```

DTO

ModelMapper (Get)

Mapiranje povratnih vrednosti metoda servisa iz objekata modela u DTO je poželjno izdvojiti u posebne metode servisa jer se upotrebljavaju na više mesta.

Type createDTO(Type source)

Collection<Type> createDTO(Collection<Model> source)

Potrebna je jedna ovakva metoda za mapiranje objekta modela na DTO i jedna za mapiranje kolekcije objekata modela na kolekciju DTO.

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;
    private final ModelMapper mapper = new ModelMapper();

    :

    private FilmDTOGet createDTO(Film film) {
        return mapper.map(film, FilmDTOGet.class);
    }

    private Collection<FilmDTOGet> createDTO(Collection<Film> filmovi) {
        Collection<FilmDTOGet> filmDTOs = new ArrayList<>();
        for (Film itFilm: filmovi) {
            FilmDTOGet filmDTO = createDTO(itFilm);
            filmDTOs.add(filmDTO);
        }
        return filmDTOs;
    }

    @Override
    public FilmDTOGet get(long id) {
        Film film = bioskop.getFilmovi().get(id);
        if (film == null) {
            throw new NoSuchElementException("Film nije pronaden!");
        }
        return createDTO(film);
    }

    @Override
    public Collection<FilmDTOGet> getAll() {
        Collection<Film> filmovi = bioskop.getFilmovi().values();
        return createDTO(filmovi);
    }

    :
}
```

Primer

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

Validacija

Java Bean Validation

Zadatak *controller*-a između ostalog je da proveriti strukturu zahteva (prisutnost i mogućnost parsiranja svih parametara). Ovo *Spring* aplikacija obavlja automatski pre pozivanja metode *controller*-a. U slučaju neispravnog zahteva nastaje izuzetak.

Zadatak servisa između ostalog je da ispitaju parametre poziva naspram unapred definisanih ograničenja i spreče modifikaciju podataka usled neispravnih vrednosti.

Moguće je izvršiti automatsku proveru parametara pre poziva metode servisa upotrebom *Java Bean Validation* API-a. U slučaju neispravnog zahteva nastaje izuzetak.

API je potrebno uvesti kroz *dependency* u datoteci *pom.xml*.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Validacija

Java Bean Validation

Parametri metoda servisa su DTO, pa je baš u njihovim klasama pogodno navesti ograničenja za svaki od atributa pojedinačno.

Postoji mnoštvo predefinisanih ograničenja: <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/>

Ograničenja se zadaju anotacijama nad atributima klase. Moguće je za isti atribut zadati više ograničenja

```
abstract class FilmDTO {  
    @Positive(message = "ID mora biti validan.", groups = {Validation.Update.class})  
    private long id;  
  
    @NotBlank(message = "Naziv ne sme biti prazan.", groups = {Validation.Add.class, Validation.Update.class})  
    private String naziv;  
  
    @Min(value = 5, message = "Trajanje ne sme biti manje od 5.", groups = {Validation.Add.class, Validation.Update.class})  
    private int trajanje;  
    :  
}  
  
public class FilmDTOAddUpdate extends FilmDTO {  
    @NotEmpty(message = "Bar jedan žanr mora biti zadat.", groups = {Validation.Add.class, Validation.Update.class})  
    private long[] zanrIds;  
    :  
}
```

Validacija

Java Bean Validation

- *message* atribut ograničenja navodi poruku kojom će biti opisan izuzetak usled neuspešne validacije

```
abstract class FilmDTO {
    @Positive(message = "ID mora biti validan.", groups = {Validation.Update.class})
    private long id;

    @NotBlank(message = "Naziv ne sme biti prazan.", groups = {Validation.Add.class, Validation.Update.class})
    private String naziv;

    @Min(value = 5, message = "Trajanje ne sme biti manje od 5.", groups = {Validation.Add.class, Validation.Update.class})
    private int trajanje;
    :
}

public class FilmDTOAddUpdate extends FilmDTO {
    @NotEmpty(message = "Bar jedan žanr mora biti zadat.", groups = {Validation.Add.class, Validation.Update.class})
    private long[] zanrIds;
    :
}
```

Validacija

Java Bean Validation

- *groups* atribut ograničenja sadrži niz klasa kojima se ono svrstava u neku grupu (npr. ograničenje važi u slučaju dodavanja ili u slučaju izmene ili u oba slučaja)

Ove klase mogu jednostavno biti i interfejsi čija je namena sa stanovišta sintakse isključivo da definišu novi tip kojim će biti konfigurisan neki drugi element sintakse (u ovom slučaju anotacija). Ovakvi interfejsi se zovu marker interfejsi.

```
public interface Validation {  
    public interface Add {}  
    public interface Update {}  
}
```

```
abstract class FilmDTO {  
    @Positive(message = "ID mora biti validan.", groups = {Validation.Update.class})  
    private long id;  
  
    @NotBlank(message = "Naziv ne sme biti prazan.", groups = {Validation.Add.class, Validation.Update.class})  
    private String naziv;  
  
    @Min(value = 5, message = "Trajanje ne sme biti manje od 5.", groups = {Validation.Add.class, Validation.Update.class})  
    private int trajanje;  
  
    :  
}
```

```
public class FilmDTOAddUpdate extends FilmDTO {  
    @NotEmpty(message = "Bar jedan žanr mora biti zadat.", groups = {Validation.Add.class, Validation.Update.class})  
    private long[] zanrIds;  
  
    :  
}
```

Validacija

Java Bean Validation

@Validated anotacija na nivou servisa govori *Spring* aplikaciji da je u nekim metodama servisa potrebno izvršiti validaciju parametara.

@Validated anotacija na nivou metode servisa govori *Spring* aplikaciji da je baš u toj metodi servisa potrebno obaviti validaciju parametara. U okviru ove anotacije se navodi i grupa ograničenja koja će važiti prilikom validacije pri pozivu te metode.

@Valid anotacija na nivou parametra metode servisa govori *Spring* aplikaciji da je baš za taj parametar metode potrebno obaviti validaciju.

```
@Service
@Validated
public class InMemoryFilmService implements FilmService {
    :

    @Override
    @Validated(Validation.Add.class)
    public void add(@Valid FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronaden nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        Film film = mapper.map(filmDTO, Film.class);
        film.setId(id);
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    @Override
    @Validated(Validation.Update.class)
    public void update(@Valid FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronaden nijedan žanr!");
        }
        long id = filmDTO.getId();
        Film film = bioskop.getFilmovi().get(id);
        mapper.map(filmDTO, film);
        film.setZanrovi(zanrovi);
    }
    :
}
```

Validacija

Java Bean Validation

```
abstract class FilmDTO {
    @Positive(
        message = "ID mora biti validan.",
        groups = {Validation.Update.class})
    private long id;

    @NotBlank(
        message = "Naziv ne sme biti prazan.",
        groups = {Validation.Add.class, Validation.Update.class})
    private String naziv;

    @Min(
        value = 5,
        message = "Trajanje ne sme biti manje od 5.",
        groups = {Validation.Add.class, Validation.Update.class})
    private int trajanje;
    :
}
```

```
public class FilmDTOAddUpdate extends FilmDTO {
    @NotEmpty(
        message = "Bar jedan žanr mora biti zadat.",
        groups = {Validation.Add.class, Validation.Update.class})
    private long[] zanrIds;
    :
}
```

```
@Service
@Validated
public class InMemoryFilmService implements FilmService {
    :

    @Override
    @Validated(Validation.Add.class)
    public void add(@Valid FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronaden nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        Film film = mapper.map(filmDTO, Film.class);
        film.setId(id);
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    @Override
    @Validated(Validation.Update.class)
    public void update(@Valid FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronaden nijedan žanr!");
        }
        long id = filmDTO.getId();
        Film film = bioskop.getFilmovi().get(id);
        mapper.map(filmDTO, film);
        film.setZanrovi(zanrovi);
    }
    :
}
```

Validacija

Java Bean Validation

```
abstract class FilmDTO {
    @Positive(
        message = "ID mora biti validan.",
        groups = {Validation.Update.class})
    private long id;

    @NotBlank(
        message = "Naziv ne sme biti prazan.",
        groups = {Validation.Add.class, Validation.Update.class})
    private String naziv;

    @Min(
        value = 5,
        message = "Trajanje ne sme biti manje od 5.",
        groups = {Validation.Add.class, Validation.Update.class})
    private int trajanje;
    :
}
```

```
public class FilmDTOAddUpdate extends FilmDTO {
    @NotEmpty(
        message = "Bar jedan žanr mora biti zadat.",
        groups = {Validation.Add.class, Validation.Update.class})
    private long[] zanrIds;
    :
}
```

```
@Service
@Validated
public class InMemoryFilmService implements FilmService {
    :

    @Override
    @Validated(Validation.Add.class)
    public void add(@Valid FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronaden nijedan žanr!");
        }
        long id = bioskop.nextFilmId();
        Film film = mapper.map(filmDTO, Film.class);
        film.setId(id);
        film.setZanrovi(zanrovi);
        bioskop.getFilmovi().put(id, film);
    }

    @Override
    @Validated(Validation.Update.class)
    public void update(@Valid FilmDTOAddUpdate filmDTO) {
        Set<Zanr> zanrovi = new LinkedHashSet<>();
        for (long itZanrId: filmDTO.getZanrIds()) {
            Zanr itZanr = bioskop.getZanrovi().get(itZanrId);
            zanrovi.add(itZanr);
        }
        if (zanrovi.isEmpty()) {
            throw new NoSuchElementException("Nije pronaden nijedan žanr!");
        }
        long id = filmDTO.getId();
        Film film = bioskop.getFilmovi().get(id);
        mapper.map(filmDTO, film);
        film.setZanrovi(zanrovi);
    }
    :
}
```

Primer

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

Servisni sloj

Prednosti

Provides separation of concern

- Service layer provides code modularity, the business logic and rules are specified in the service layer which in turn calls DAO layer, the DAO layer is then only responsible for interacting with DB.

Provides Security

- If you provide a service layer that has no relation to the DB, then it is more difficult to gain access to the DB from the client except through the service. If the DB cannot be accessed directly from the client (and there is no trivial DAO module acting as the service) then an attacker who has taken over the client cannot have access to your data directly.

Provide Loose Coupling

- Service layer can also be used to serve loose coupling in the application. Suppose your controller has 50 methods and in turn it calls 20 Dao methods, now at later point you decide to change the Dao methods serving these controllers. You need to change all the 50 methods in controller. Instead if you have 20 service methods calling those 20 Dao methods, you need to make change in only 20 Service methods to point to a new Dao.

Pretrage

Forma bez *action* atributa šalje zahtev na trenutni URL.

Forma bez *method* atributa šalje GET zahtev.

Tabela sme da bude ugnježdena u formu ali ne i obrnuto.

```
<form>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Redni broj</th>
        <th>Naziv</th>
        <th>Žanr</th>
        <th>Trajanje</th>
        <th></th>
      </tr>
      <tr class="align-middle">
        <th></th>
        <th><input type="text" class="form-control" name="naziv"></th>
        <th><select class="form-select" name="zanrId">
          :
        </select>
        <th></th>
        <th><div class="input-group">
          <span class="input-group-text">od</span>
          <input type="number" class="form-control" name="trajanjeOd">
        </div>
        <div class="input-group">
          <span class="input-group-text">do</span>
          <input type="number" class="form-control" name="trajanjeDo">
        </div>
        <th></th>
        <th><button type="submit" class="btn btn-primary">Pretraži</button></th>
      </tr>
    </thead>
  </table>
</form>
```

Redni broj	Naziv	Žanr	Trajanje
1	Avengers: Endgame	<ul style="list-style-type: none">naučna fantastikaakcijaavantura	182 projekcije
2	Life	<ul style="list-style-type: none">naučna fantastikahoror	110 projekcije
3	It: Chapter 2	<ul style="list-style-type: none">horor	170 projekcije
4	Pirates of the Caribbean: Dead Men Tell No Tales	<ul style="list-style-type: none">akcijakomedijaavantura	153 projekcije



Pretrage

Ista metoda *controller*-a mora da odgovori i na zahtev bez parametara (pri redovnom prikazu stranice) i na zahtev sa parametrima (u slučaju pretrage).

Parametri moraju biti **opcionni**.

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;

    @GetMapping("")
    public String getAll(ModelMap request,
        @RequestParam(required = false, defaultValue = "") String naziv,
        @RequestParam(required = false, defaultValue = "0") long zanrId,
        @RequestParam(required = false, defaultValue = "0") int trajanjeOd,
        @RequestParam(required = false, defaultValue = "0") int trajanjeDo) {
        request.addAttribute("filmovi", filmService.get(naziv, zanrId, trajanjeOd, trajanjeDo));
        request.addAttribute("zanrovi", zanrService.getAll());
        return "filmovi"; // forwarding na template
    }
}
```



Pretrage

```
@Service
public class InMemoryFilmService implements FilmService {
    private final Bioskop bioskop;
    private final ModelMapper mapper = new ModelMapper();



    :

    @Override
    public Collection<FilmDTOGet> get(String naziv, long zanrId, int trajanjeOd, int trajanjeDo) {
        Collection<Film> filmovi = bioskop.getFilmovi().values();

        List<Film> rezultat = new ArrayList<>();
        for (Film itFilm: filmovi) {
            if ((naziv == null || itFilm.getNaziv().toLowerCase().contains(naziv.toLowerCase())) &&
                (zanrId <= 0 || itFilm.getZanr(zanrId) != null) &&
                (trajanjeOd <= 0 || itFilm.getTrajanje() >= trajanjeOd) &&
                (trajanjeDo <= 0 || itFilm.getTrajanje() <= trajanjeDo)) {
                rezultat.add(itFilm);
            }
        }
        return createDTO(rezultat);
    }

    :

}
```



Pretrage

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;

    @GetMapping("")
    public String getAll(ModelMap request,
        @RequestParam(required = false, defaultValue = "") String naziv,
        @RequestParam(required = false, defaultValue = "0") long zanrId,
        @RequestParam(required = false, defaultValue = "0") int trajanjeOd,
        @RequestParam(required = false, defaultValue = "0") int trajanjeDo) {
        request.addAttribute("filmovi", filmService.get(naziv, zanrId, trajanjeOd, trajanjeDo));
        request.addAttribute("zanrovi", zanrService.getAll());
        return "filmovi"; // forwarding na template
    }
}
```

Pretrage

U novogenerisanoj stranici ne ostaju vrednosti koje su bile unesene u polja forme.

```
<form>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Redni broj</th>
        <th>Naziv</th>
        <th>Žanr</th>
        <th>Trajanje</th>
        <th></th>
      </tr>
      <tr class="align-middle">
        <th></th>
        <th><input type="text" class="form-control" name="naziv"></th>
        <th><select class="form-select" name="zanrId">
          :
        </select>
        <th>
          <div class="input-group">
            <span class="input-group-text">od</span>
            <input type="number" class="form-control" name="trajanjeOd">
          </div>
          <div class="input-group">
            <span class="input-group-text">do</span>
            <input type="number" class="form-control" name="trajanjeDo">
          </div>
        </th>
        <th><button type="submit" class="btn btn-primary">Pretraži</button></th>
      </tr>
    </thead>
    :
  </table>
</form>
```

Filmovi

localhost:8080/filmovi?naziv=er&zanrId=0&trajanjeOd=150&trajanjeDo=180

Početna Žanrovi Filmovi Projekcije Korisnici

Prikaz svih Dodavanje

Redni broj	Naziv	Žanr	Trajanje
	<input type="text"/>	svi	od <input type="text"/> do <input type="text"/>
1	It: Chapter 2	• horor	170 projekcije

Pretrage

U novogenerisanoj stranici ne ostaju vrednosti koje su bile unesene u polja forme.

Vrednosti međutim ostaju u URL-u zahteva.

Moguće ih je prepisati na novu stranicu iz URL-a posredstvom predefinisano *param* objekta koji postoji u svakom *template*-u. Parametri se čitaju *po nazivu*.

```
<form>
<table class="table table-striped">
  <thead>
    <tr>
      <th>Redni broj</th>
      <th>Naziv</th>
      <th>Žanr</th>
      <th>Trajanje</th>
    </tr>
    <tr class="align-middle">
      <th></th>
      <th><input type="text" class="form-control" name="naziv" th:value="${param.naziv}"></th>
      <th><select class="form-select" name="zanrId">
        :
      </select>
      <th>
        <div class="input-group">
          <span class="input-group-text">od</span>
          <input type="number" class="form-control" name="trajanjeOd" th:value="${param.trajanjeOd}">
        </div>
        <div class="input-group">
          <span class="input-group-text">do</span>
          <input type="number" class="form-control" name="trajanjeDo" th:value="${param.trajanjeDo}">
        </div>
      </th>
      <th><button type="submit" class="btn btn-primary">Pretraži</button></th>
    </tr>
  </thead>
  :
</table>
</form>
```

Search results table:

Redni broj	Naziv	Žanr	Trajanje
1	It: Chapter 2	• horor	170 projekcije

Primer

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

Zadatak 1

Po ugledu na primer *com.ftninformatika.jwd.modul2.termin6.bioskop* implementirati servisni sloj u zadatku *com.ftninformatika.jwd.modul2.termin6.dostava*:

1. Definirati DTO klase za sve tipove entiteta u paketu *dto*:
 - i. apstraktnu DTO klasu koja sadrži sve deljene attribute
 - ii. konkretnu DTO klasu za povratne vrednosti *get* metoda servisa
 - iii. Konkretnu DTO klasu za parametre *add* i *update* metoda servisa
2. Definirati interfejse servisa u paketu *service*
3. Započeti *in-memory* implementacije interfejsa u paketu *service.impl*
4. *Inject*-ovati klasu *Dostava* u započete implementacije servisa
5. *Inject*-ovati započete implementacije servisa u *controller*-e uz oslonac na njihove interfejse
6. Uvesti *ModelMapper dependency* u *pom.xml* datoteku
7. Definirati *ModelMapper* atribut i *createDTO(...)* metode servisa
8. Implementirati jednu po jednu metodu jednog po jednog servisa, pozvati je u odgovarajućoj metodi *controller*-a i testirati funkcionalnost

Zadatak 2

Po ugledu na primer *com.ftninformatika.jwd.modul2.termin6.bioskop* dopuniti rešenje zadatka 1 tako da vrši validaciju parametara metoda servisa uz oslonac na *Java Bean Validation API*:

1. Uvesti *Java Bean Validation API dependency* u *pom.xml* datoteku
2. U paketu *dto.validation* definisati *Validation* interfejs i u njemu grupe ograničenja *Add* i *Update*
3. Definisati ograničenja u DTO klasama
4. Uključiti automatsku validaciju u *add* i *update* metodama servisa
5. Testirati validaciju

Zadatak 3

Po ugledu na primer *com.ftninformatika.jwd.modul2.termin6.bioskop* dopuniti rešenje zadatka 2 tako da uz prikaz omogući i pretragu za svaki tip entiteta:

1. Dopuniti *template*-e stranica za prikaz formama za pretragu
2. Dopuniti metode kontrolera opcionim parametrima za pretragu
3. Definisati metode servisa *get(...)* i u njima implementirati pretrage
4. U metodama kontrolera pozvati implementirane metode umesto *getAll()* metoda
5. Dopuniti *template*-e stranica za prikaz tako da zadrže vrednost poslatih parametara za pretragu