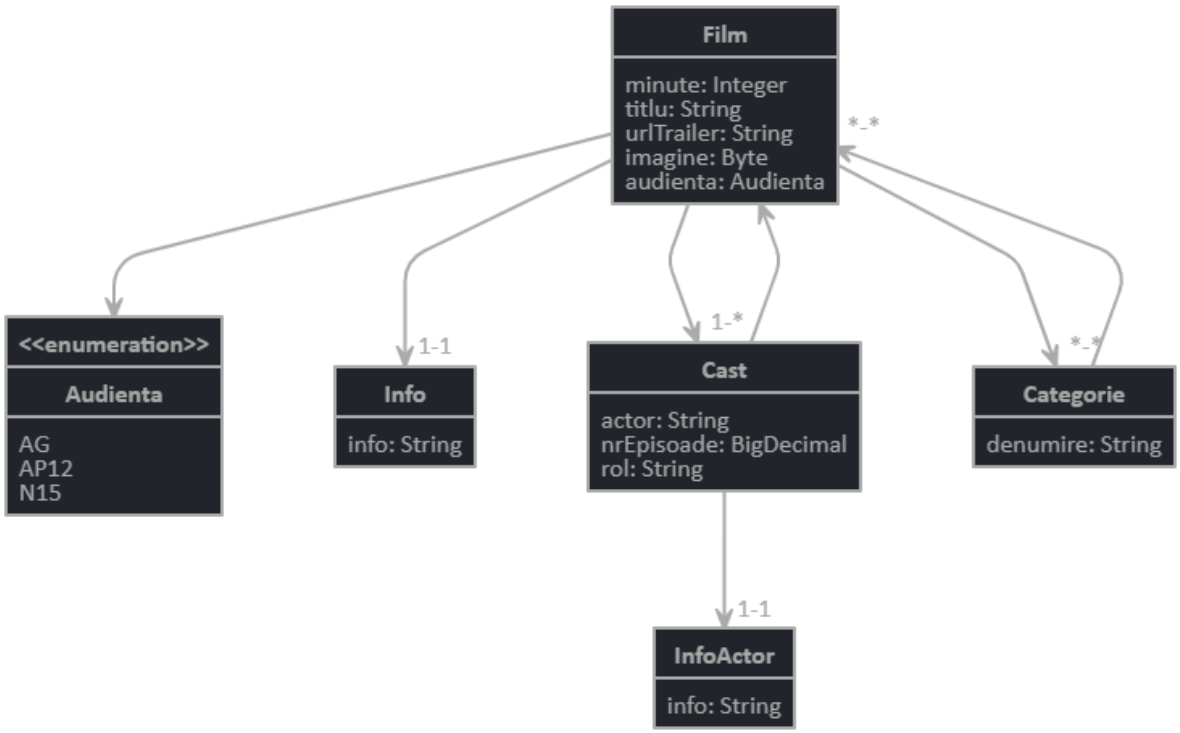
Laborator 3: JPA, Java Persistence API

RELATII

Analizati relatiile dintre urmatoarele entitati:



@Entity O entitate este o clasa java ale carei instante sunt memorate intr-o baza de date relationala.
javax.persistence.EntityManager controleaza ciclul de viata al entitatilor dintr-un **persistence context**. Dupa creare entitatile pot fi, pe durata unei sesiuni:

ManagedEntity orice modificare din sesiunea curenta va fi propagata in baza de date, entitatile sunt asociate unui *persistence context*.
DetachedEntity modificarile nu sunt salvate, fie sesiunea a fost inchisa, fie entitatile au fost eliminate din *persistence context*, spre exemplu prin apelarea metodei *evict*.
RemovedEntity entitatile sunt asociate unui *persistence context* si urmeaza sa fie sterse din baza de date.

@OneToOne O entitate este asociata unei singure alte entitati.
@OneToMany O entitate este asociata mai multor entitati care pot fi memorate intr-o colectie de tip List, Set, Map, SortedSet, SortedMap etc.
@ManyToOne Este inversa relatiei de tip @OneToMany.
@ManyToMany este definita prin utilizarea unui tabel asociativ. Exemplu: PRODUS – CATEGORIE.

Relatiile pot fi unidirectionale sau bidirectionale.
Intr-o relatie de tip *OneToMany* tabelul corespunzator entitatilor ***many*** retine cheia externa a relatiei.

mappedBy defineste campul care retine cheie externa.

Fetch Types:
Lazy Fetch: default pentru OneToMany si ManyToMany
Eager Fetch: default pentru ManyToOne, OneToOne

Cascade Types specifica modul in care sunt propagate modificarile de la o entitate parinte la entitatile fiu.
DETACH
MERGE
PERSIST
REFRESH
REMOVE
ALL

1. Importati in IntelliJ proiectul laborator3. Din meniu selectati File – New projrct from existing sources. Rulati mvn clean install.

Clasele reprezentand entitati vor fi adaugate in pachetul **domain**. In alte proiecte denumirea uzuala pentru pachetul care contine clase reprezentatand entitati este **model**.

2. Generati folosind IntelliJ metode de tip setter si getter pentru toate atributele claselor Film si Info. Din meniu selectati Code – Generate.
3. Adnotati clasele Film si Info cu @Entity.
4. Adaugati in clasa Film atributul **info** care va modela o relatie de tip OneToOne. Adaugati in clasa Info atributul **film**. Entitatea *owner* va fi film. Vor putea fi sterse entitati de tip Info fara a fi sterse entitatile de tip Film asociate, daca insa vom sterge o entitate de tip film, va fi stearsa si entitatea de tip Info corespunzatoare.

```
@OneToOne(cascade = CascadeType.ALL)
private Info info;
```

```
@OneToOne
private Film film;
```

5. Adaugati in clasele Film si Info atribute de tip Id.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

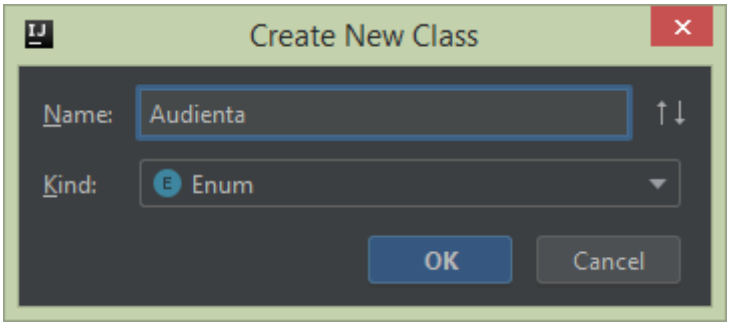
6. Adaugati metode de tip setter si getter pentru atributele film, info si id.
7. Adnotati cu @Lob atribute info si imagine. Coloanele asociate acestor atribute vor fi create in baza de date cu tipul CLOB si BLOB

```
@Lob
private String info;

@Lob
private Byte[] imagine;
```

8. Adaugati in clasa Cast, atributele id si film si metode de tip setter si getter. Adnotati clasa Cast cu @Entity.
9. Adaugati in clasa Film atributul cast. Entitatea *owner* pentru relatie va fi film.

```
@ManyToOne
private Film film;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "film")
private Set<Cast> cast;
```

10. Creati o relatie de tip OneToOne intre Cast si InfoActor. Relatia va fi creata astfel incat daca este stearsa o inregistrare din Cast inregistrarea corespunzatoare din InfoActor sa nu fie stearsa. Relatia va fi unidirectionala.
11. Adaugati in pachetul domain clasa enum Audienta. Adaugati in Film un atribut de tipul Audienta.



```
@Enumerated(value = EnumType.STRING)
private Audienta audienta;
```

EnumType.STRING in baza de date se vor memora string-uri.
EnumType.ORDINAL valoare default. in baza de date se vor memora numere de ordine.

12. Adaugati in clasa categorie atributul filme.

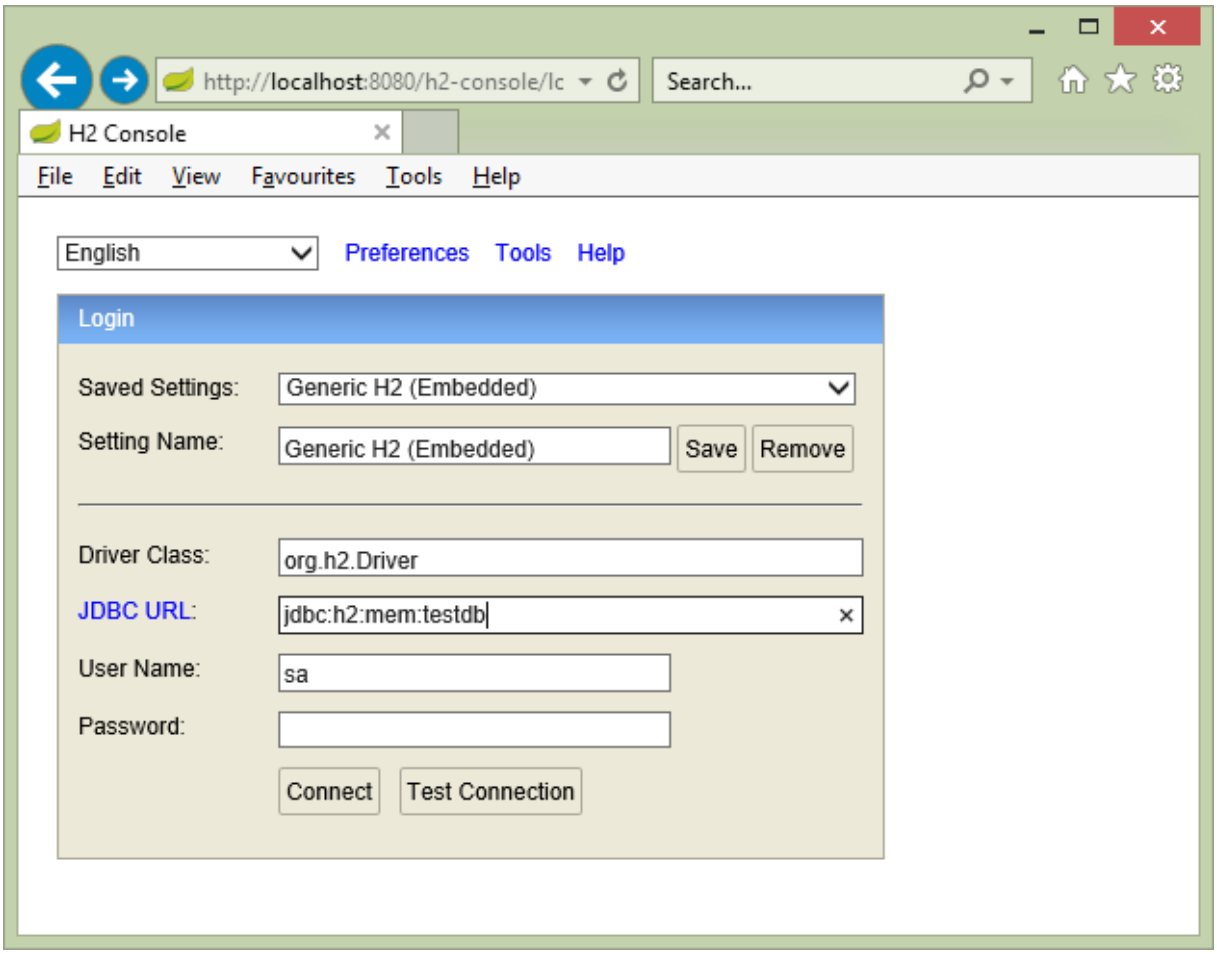
```
@Enumerated(value = EnumType.STRING)
private Audienta audienta;
```

13. Executati aplicatia cu spring-boot:run. Testati in browser urmatoarele url-uri si conectati-va la baza de date H2.

```
http://localhost:8080/

http://localhost:8080/h2-console
```

H2 este o baza de date *in memory* (**IMDB**).
Pentru integrarea acesteia in aplicatiile spring se va aduaga in pom dependenta:

```
<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
</dependency>
```



14. Adaugati o relatie de tip ManyToMany. In clasa Film adaugati atributul categorii. Reporniti aplicatia si testati crearea tabelului asociativ.

```
@ManyToMany(mappedBy = "categorii")
private Set<Film> filme;
```

```
@ManyToMany
@JoinTable(name = "film_categorie",
        joinColumns = @JoinColumn(name = "film_id"),
        inverseJoinColumns = @JoinColumn(name = "categorie_id"))
private Set<Categorie> categorii;
```

spring.jpa.hibernate.dll-auto

none
validate recomandat in productie, previne pornirea aplicatiei daca entitatile jpa nu corespund schemei bazei de date
update va modifica schema conform entitatilor jpa
create va crea baza de date
create-drop va crea baza de date iar la shutdown baza de date va fi stearsa

Spring-boot va folosi default optiunea *create-drop*

Daca se utilizeaza create sau create-drop fisierul import.sql din classpath poate fi utilizat pentru executarea unor comenzi ddl suplimentare si pentru comenzi dml pentru popularea bazei de date.

Alternativ, Spring DataSource initializer, configurat cu Spring Boot va executa default fisierele schema.sql si data.sql pentru comenzi ddl respectiv dml.

15. Creati fisierul data.sql in folderul resources. Executati aplicatia si testati in consola H2 continutul tabelului CATEGORIE si FILM.

```
insert into film (titlu) values ('Mary Poppins Returns');
insert into film (titlu) values ('Grinch');
insert into categorie (denumire) values ('animatie');
insert into categorie (denumire) values ('fantasy');
insert into categorie (denumire) values ('comedie');
```

```
select * from categorie
```

SPRING DATA REPOSITORIES

CrudRepository <entity, id > primeste ca tipuri generice tipul entitatii pentru care va defini operatii CRUD si tipul atributului Id al entitatii.
Denumirea interfetelor care extind Repository este prin conventie formata din numele entitatii pentru care este creat un repository si sufixul Repository.

16. Creati o interfata de tip repository pentru clasele Film, Categorie si InfoActor.

```
package apbdoo.laboratorul3.repositories;

import apbdoo.laboratorul3.domain.Film;
import org.springframework.data.repository.CrudRepository;

public interface FilmRepository
                    extends CrudRepository<Film, Long> {
}
```

17. Adaugati in toate entitatile initializari pentru atributele de tip set sau lista.

```
private Set<Film> filme = new HashSet<Film>();
```

18. Adaugati in pachetul apbdoo.laboratorul3 clasa GenerareFilme. Adaugati instructiunile import necesare si un constructor.

```
package apbdoo.laboratorul3;

@Component
public class GenerareFilme implements
ApplicationListener<ContextRefreshedEvent> {
    private final FilmRepository filmRepository;
    private final InfoActorRepository infoActorRepository;
    private final CategorieRepository categorieRepository;


    @Override
    public void onApplicationEvent(ContextRefreshedEvent
contextRefreshedEvent) {
        filmRepository.saveAll(getFilme());
    }

    private List<Film> getFilme(){
        List<Film> filme = new ArrayList<Film>();
        Film film = new Film();
        film.setAudienta(Audienta.AG);
        film.setMinute(120);
        film.setTitlu("Aquaman");
        filme.add(film);
       return filme;
    }
}
```

19. Afisati in consola H2 continutul tabelului film.
20. Adaugati filmului “Aquaman” categoria “fantasy”. Folositi metoda findByDenumire a interfetei CategorieRepository.

```
public interface CategorieRepository extends
CrudRepository<Categorie, Long> {
    Categorie findByDenumire(String denumire);
}
```

21. Afisati in consola H2 continutul tabelului film_categorie.
22. Modificati metoda setInfo a clasei Film:

```
public void setInfo(Info info) {
        this.info = info;
```

```
        info.setFilm(this);
}
```

23. In clasa GenerareFilme adaugati informatii despre filmul *Aquaman*.

```
Info info = new Info();
info.setInfo("Arthur Curry learns that" +
                " he is the heir to the underwater " +
                "kingdom of Atlantis, and must step " +
                "forward to lead his people and be a hero " +
                "to the world.");
film.setInfo(info);
```

24. Adaugati in clasa InfoActor atributul nume si in InfoActorRepository o metoda care sa caute informatii despre un actor primind ca argument un nume de actor.

```
public interface InfoActorRepository extends
CrudRepository<InfoActor, Long> {
    InfoActor findByNume(String nume);
}
```

25. Adaugati in data.sql instructiuni insert pentru a adauga linii in tabelul InfoActor.

```
insert into info_actor(nume, info) values ('Jason Momoa',
'Joseph Jason Namakaeha Momoa was born on Aug1,79,in Hawaii. ')
```

26. Adaugati in clasa Film metoda addCast(Cast cast)

```
public void addCast(Cast cast){
        this.cast.add(cast);
        cast.setFilm(this);
    }
```

27. In clasa GenerareFilme adaugati in tabelul cast o linie care sa corespunda filmului *Aquaman.*

```
public void addCast(Cast cast){
        this.cast.add(cast);
        cast.setFilm(this);
    }
```

28. Afisati in consola H2 continutul tabelelor din schema.
29. Adaugati o interfata FilmService si o implementare a acesteia FilmServiceImpl. Clasele vor fi adaugate intr-un nou pachet denumit services.

```
public interface FilmService {
    Set<Film> getFilme();
}
```

```
@Service
public class FilmServiceImpl implements FilmService{
    FilmRepository filmRepository;

    @Override
    public Set<Film> getFilme() {
        Set<Film> filme = new HashSet<Film>();

filmRepository.findAll().iterator().forEachRemaining(filme::add);
        return filme;
    }
}
```

30. Adaugati in clasa IndexController metoda getFilmePage.

```
@RequestMapping({"/filme"})
    public String getFilmePage(Model model){

        model.addAttribute("filme", filmService.getFilme());

        return "filme";
    }
```

31. Adaugati in serviciul FilmService o metoda pentru regasirea unui film in functie de un id primit ca argument.

```
Film findById(Long l);
```

```
@Override
    public Film findById(Long l) {

        Optional<Film> filmOptional = filmRepository.findById(l);

        if (!filmOptional.isPresent()) {
            throw new RuntimeException("Movie not found!");
        }

        return filmOptional.get();
    }
```

32. Adaugati un controller care sa raspunda unor requesturi de tipul /film/info/id.

```
@RequestMapping("/film/info/{id}")
    public String showById(@PathVariable String id, Model model){

        model.addAttribute("film", filmService.findById(new
Long(id)));

        return "info";
    }
```

33. Adaugati in serviciul FilmService o metoda pentru inregistrarea/salvarea unui obiect de tipul Film.

```
@Override
    public Film saveFilm(Film film) {
        Film savedFilm = filmRepository.save(film);
        return savedFilm;
    }
```

34. Adaugati in controller metode care sa raspunda unor requesturi de tipu /film/new:

```
@RequestMapping("/film/new")
    public String newFilm(Model model){
        model.addAttribute("film", new Film());

        return "new";
    }


@PostMapping("film")
    public String saveOrUpdate(@ModelAttribute Film film){
        Film savedCommand = filmService.saveFilm(film);

        return "redirect:/film/info/" + film.getId();
    }
```

Tymeleaf Java Template Engine folosit pentru generarea de fisiere html, xml, html5. Spre deosebire de fisierele JSP, templateurile tymeleaf pot fi vizualizate direct in browser.