Agora vamos salvar os dados de episodios no modelo relacional.

Em vez de buscar os dados na web, vamos buscar os dados das series que já estão no nosso banco de dados.

Primeiro vamos criar um atributo que seja para classe principal onde vamos guardar as series

* Precisamos fazer isso porque vamos usar essa lista em mais de um local de nossa classe.

private List<Serie> series = new ArrayList<>();

Primeiro vamos exibir as series disponíveis em nosso banco de dados.

private void buscarEpisodioPorSerie(){  
  
 try {  
 buscarListaDeSeries(); // Exibir lista de series

**Perguntar qual serie o usuário quer ver os episódios.**

System.*out*.println("Escolha uma série pelo nome: ");  
 var nomeSerie = leitura.nextLine();

**Filtrar pelo nome, e atribuir os dados em um Optinal**

Optional<Serie> serie = series.stream()  
 .filter(s -> s.getTitulo().toLowerCase().contains(nomeSerie.toLowerCase()))  
 .findFirst();

**Agora devemos verificar se existe.**

if (serie.isPresent()) {  
 var serieEncontrada = serie.get();  
  
 List<DadosTemporada> temporadas = new ArrayList<>();

Se existir vamos percorrer a lista como padrão.

Integer totalTemporadas = serieEncontrada.getTotalTemporadas();  
if (totalTemporadas == null) {  
 System.*out*.println("Titulo não é uma serie ou não tem temporadas");  
} else {  
 for (int i = 1; i <= serieEncontrada.getTotalTemporadas(); i++) {  
 var json = consumo.obterDados(ENDERECO + serieEncontrada.getTitulo().replace(" ", "+") + "&season=" + i + API\_KEY);  
 DadosTemporada dadosTemporada = conversor.obterDados(json, DadosTemporada.class);  
 temporadas.add(dadosTemporada);  
 }  
}  
temporadas.forEach(System.*out*::println);

List<Episodio> episodios = temporadas.stream()  
 .flatMap(d -> d.episodios().stream()  
 .map(e -> new Episodio(d.numero(), e)))  
 .collect(Collectors.*toList*());  
serieEncontrada.setEpisodios(episodios);  
repository.save(serieEncontrada);

Este código faz o seguinte:

1. **Iteração sobre as temporadas**:
   * temporadas.stream() cria um fluxo (stream) da lista de temporadas.
2. **Desdobramento dos episódios de cada temporada**:
   * flatMap(d -> d.episodios().stream()...) aplica a função fornecida a cada temporada (d), onde d.episodios().stream() cria um fluxo de episódios para a temporada d.
3. **Mapeamento dos episódios para criar novos objetos Episodio**:
   * map(e -> new Episodio(d.numero(), e)) transforma cada episódio (e) em um novo objeto Episodio, passando o número da temporada (d.numero()) e o episódio (e) para o construtor de Episodio.
4. **Coleta dos resultados em uma lista**:
   * collect(Collectors.toList()) coleta todos os objetos Episodio gerados pelo mapeamento em uma lista.
5. **Configuração e salvamento da série**:
   * serieEncontrada.setEpisodios(episodios) define a lista de episódios criados na série encontrada.
   * repository.save(serieEncontrada) salva a série atualizada no repositório.

Agora dentro de Serie precisamos informar a chave estrangeira para cada episodio então vamos passar os parâmetros dentro do set.

public void setEpisodios(List<Episodio> episodios) {  
 episodios.forEach(e -> e.setSerie(this));  
 this.episodios = episodios;  
}

Aqui temos que fazer isso pq precisamos associar cada episodio a uma serie.