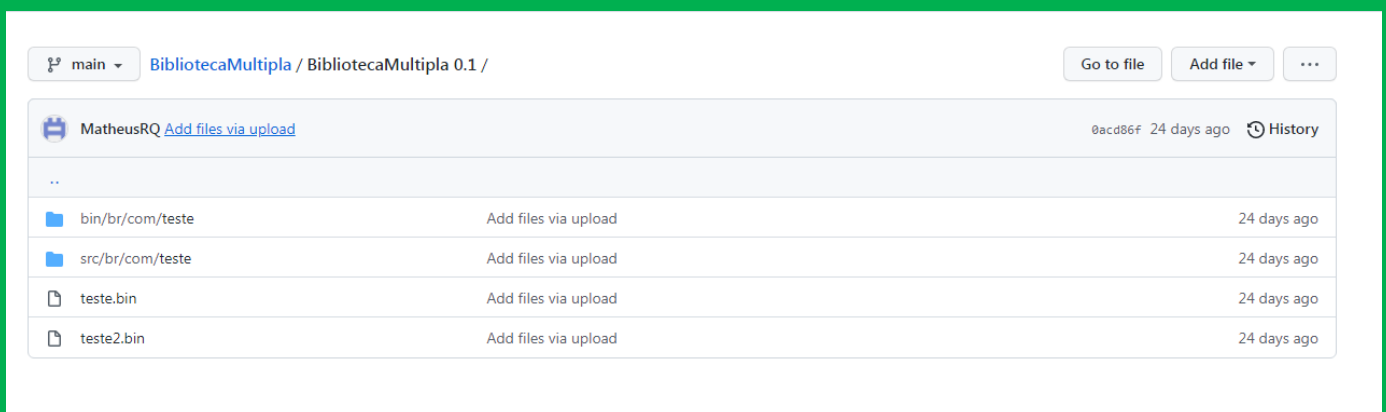


Biblioteca Múltipla.

A inspiração para esse mini projeto veio do site chamado (<https://myanimelist.net/>). Nesse site, os usuários podem, não só ver informações das animações que gostam, como também podem debater sobre e até darem notas e escreverem opiniões sobre a obra.

Baseado nessa premissa, decidi tentar criar a minha própria versão dessa ideia, porém trabalhando não apenas com animações orientais, mas com qualquer categoria de mídia possível, usando os conhecimentos adquiridos pela formação “Java e Orientação a Objetos” da Alura.

Comecei criando as classes do projeto, que seriam para representar animações orientais (com o nome de “Animes”) e animações ocidentais; criei uma classe para trabalhar com listas; decidi fazer todas as classes serem serializáveis, para eu poder trabalhar com listas salvas, não perdendo tempo recriando elas a cada teste; e, por fim, criei uma classe de JUnit para eu poder realizar todos os testes necessários envolvendo o código. Essa foi a versão 0.1 do projeto.



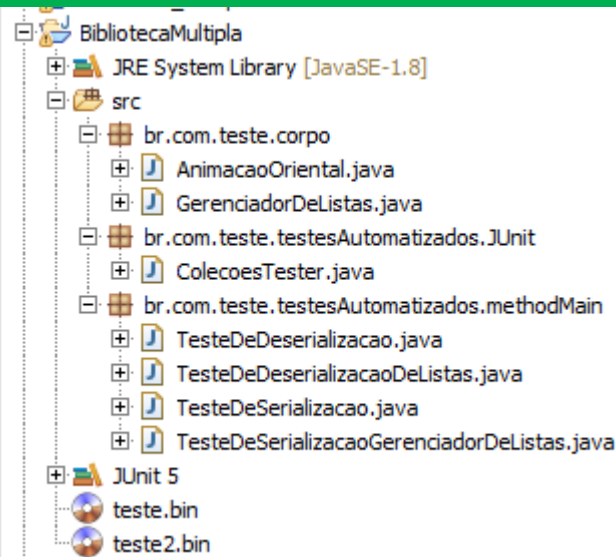
BibliotecaMultipla

Arquivos do projeto "BibliotecaMultipla".

Versão 0.1

Criação das classes "Animes", "GerenciadorDeListas"; criado teste automatizado por método main da serialização e deserialização da classe "Animes"; criação da classe "ColecoesTester", com um teste em JUnit para testar o retorno de uma Unmodifiablecollection e ver se ela soltaria uma Exception ao tentar adicionar itens nela.

(<https://github.com/MatheusRQ/BibliotecaMultipla/tree/main/BibliotecaMultipla%200.1>)



```
package br.com.teste.corpo;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 * Representa o corpo de uma animação japonesa na BibliotecaMultipla
 *
 * @author Matheus
 * @version 0.1
 */
public class AnimacaoOriental implements Serializable {

    private static final long serialVersionUID = 1L;
    private String nome;
    private int numeroDeEpisodios;
    private int ano;
    private String genero;
    private LocalDateTime dataDeRegistro = LocalDateTime.now();
    private String dataDeRegistroConvertido;

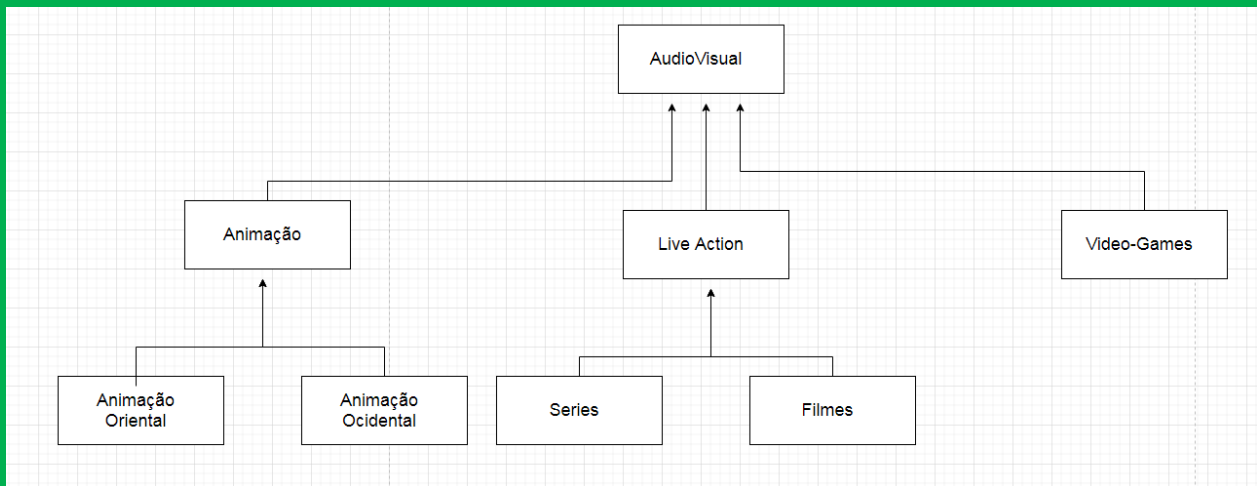
    public AnimacaoOriental(String nome, int numeroDeEpisodios, String genero, int ano) {
        if (nome == null || numeroDeEpisodios == 0) {
            throw new NullPointerException("Nome e numero de episodios sao informacoes fundamentais!");
        }
        this.nome = nome;
        this.numeroDeEpisodios = numeroDeEpisodios;
        this.ano = ano;
        this.genero = genero;

        DateTimeFormatter formatador = DateTimeFormatter.ofPattern("dd/MM/yyyy hh:mm");
        this.dataDeRegistroConvertido = formatador.format(dataDeRegistro);
    }

    @Override
    public String toString() {
        return "[Nome: " + this.nome + ", episodios: " + this.numeroDeEpisodios + ", genero: "
            + this.genero + ", ano: " + this.ano + ", registro: " + dataDeRegistroConvertido;
    }
}
```

Na versão 0.2, eu decidi mudar o nome de “Anime” para “Animacao_Oriental” (o print acima é um intermédio dos 2), além de criar a classe “AudioVisual”, que seria responsável por ser a mãe de toda a hierarquia de classes áudios visuais. Também criei uma classe no JUnit para testar a exportação de serializáveis, que ficaria separada da classe de JUnit pré-existente.

Hierarquia das Classes:



BibliotecaMultipla

Arquivos do projeto "BibliotecaMultipla".

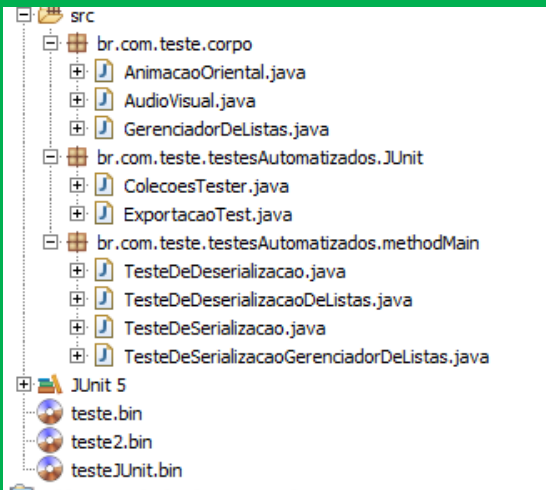
Versão 0.1

Criação das classes "Animes", "GerenciadorDeListas"; criado teste automatizado por método main da serialização e deserialização da classe "Animes"; criação da classe "ColecoesTester", com um teste em JUnit para testar o retorno de uma Unmodifiablecollection e ver se ela soltaria uma Exception ao tentar adicionar itens nela.

Versão 0.2

Mudança do nome da classe "Animes" para "AnimaçãoOriental"; criação da classe "AudioVisual" como o topo da hierarquia de classes; criação da classe "ExportacaoTest", com métodos em JUnit para assegurar a serialização e deserialização das classes do projeto.

(<https://github.com/MatheusRQ/BibliotecaMultipla/tree/main/BibliotecaMultipla%200.2>)



```
package br.com.teste.corpo;

import java.io.Serializable;

/**
 * Representa todo elemento global para Audio-visual
 *
 * @author Matheus
 * @version 0.1
 */
public class AudioVisual implements Serializable {

    private static final long serialVersionUID = 1L;
    private String nome;
    private int anoDeLancamento;
    private String genero;
    private LocalDateTime dataDeRegistro = LocalDateTime.now();
    private String dataDeRegistroConvertido;

    public AudioVisual(String nome, int anoDeLancamento, String genero) {
        if (nome == null) {
            throw new NullPointerException("Nome é uma informação fundamental!");
        }

        this.nome = nome;
        this.anoDeLancamento = anoDeLancamento;
        this.genero = genero;

        DateTimeFormatter formatador = DateTimeFormatter.ofPattern("dd/MM/yyyy hh:mm");
        this.dataDeRegistroConvertido = formatador.format(dataDeRegistro);
    }

    public String getNome() {
        return nome;
    }

    public int getAnoDeLancamento() {
        return anoDeLancamento;
    }

    public String getGenero() {
        return genero;
    }

    public String getDataDeRegistroConvertido() {
        return dataDeRegistroConvertido;
    }
}
```

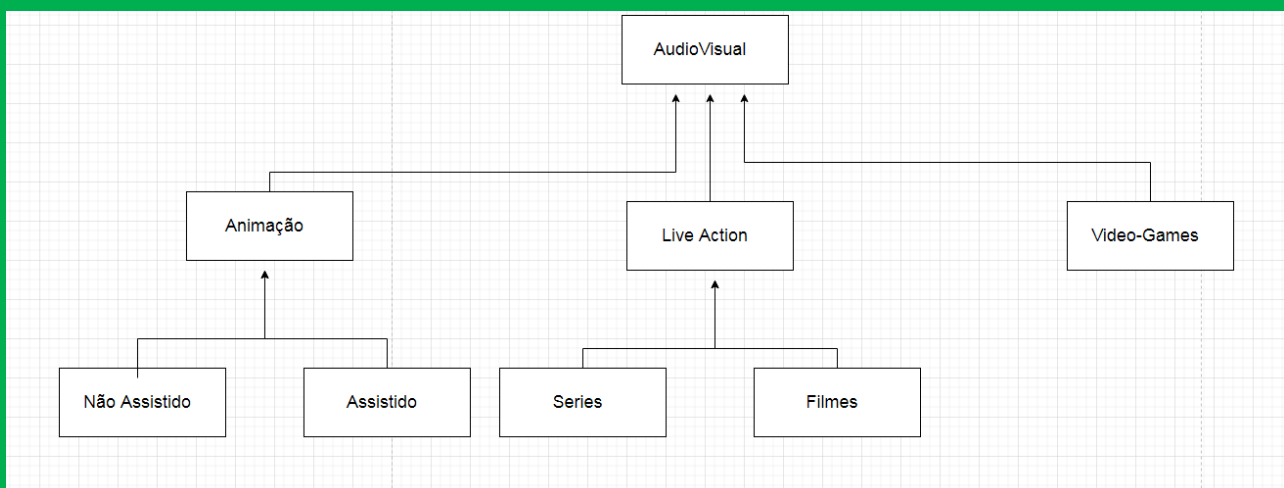
A partir desse momento, comecei a notar a dificuldade que eu teria de lidar. Trabalhar apenas com serializáveis me forçaria a trabalhar com mais listas do que eu realmente gostaria. Pensando um pouco, cheguei à conclusão que eu deveria usar os conhecimentos que adquiri na formação “SQL com MySQL Server da Oracle” da Alura.

Então fui em uma coleção de cursos, sugerida por um dos participantes da semana Java que a Alura fez, e fiz o curso de JDBC, para juntar ambas as formações nesse projeto.

Após fazê-lo, percebi que eu teria de mexer em muitas coisas na estrutura do projeto para incluir o JDBC. Entendendo minhas limitações atuais, eu decidi por começar do zero, usando o que eu já criara de modelo para o que precisasse enquanto criava tudo em volta do JDBC.

Aproveitei e repensei o modelo da hierarquia, e decidi que toda a animação, independente da nacionalidade, deveria ser tratada pela mesma classe. Porém, eu decidi criar 2 subclasses, uma para não assistidas e outras para assistidas. A motivação disso foi simples, ambas as subclasses teriam atributos diferentes; no banco de dados, para uma animação ser da categoria “não assistida”, ela teria que ter todos os atributos de uma classe de categoria “assistida” vazios, e isso seria igual para o projeto no Java. Para ir contra isso, decidi separar ambos. Essa foi a versão 0.1 pós 17/09.

Nova Hierarquia:

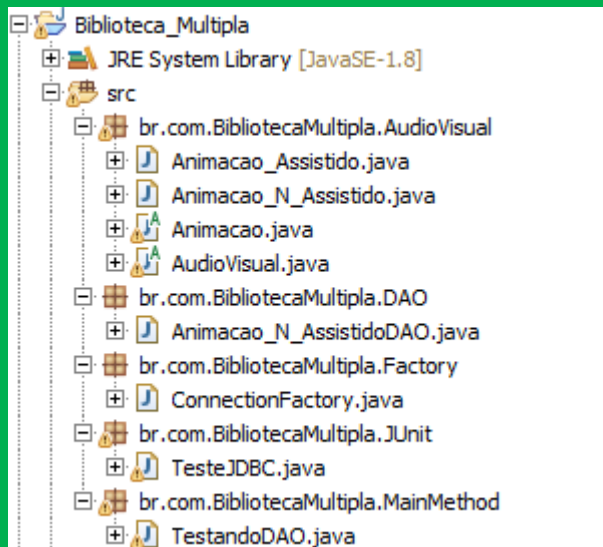


Pós 17-09

Versão 0.1

Recriação de todo o projeto, com base na utilização do JDBC; criação da classe "FactoryConnection".

(https://github.com/MatheusRO/BibliotecaMultipla/tree/main/Biblioteca_Multipla%2017-09)



```
package br.com.BibliotecaMultipla.Factory;

import java.sql.Connection;
import java.sql.SQLException;

import javax.sql.DataSource;

import com.mchange.v2.c3p0.ComboPooledDataSource;

public class ConnectionFactory {

    private DataSource dataSource;

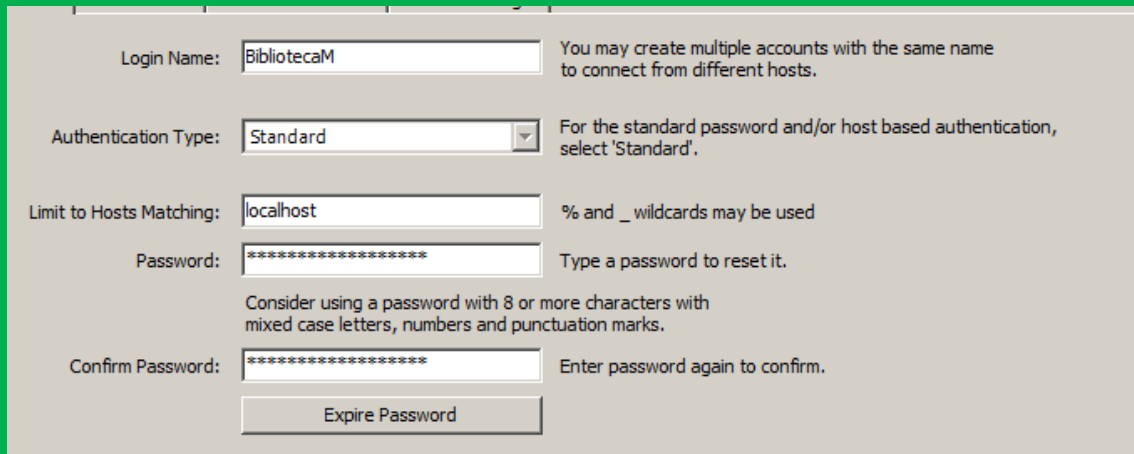
    public ConnectionFactory() {
        ComboPooledDataSource cpds = new ComboPooledDataSource();
        cpds.setJdbcUrl("jdbc:mysql://localhost/biblioteca_multipla?useTimezone=true&serverTimezone=UTC");
        cpds.setUser("???");
        cpds.setPassword("???");

        this.dataSource = cpds;
    }

    public Connection recuperarConexao() {
        try {
            return this.dataSource.getConnection();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

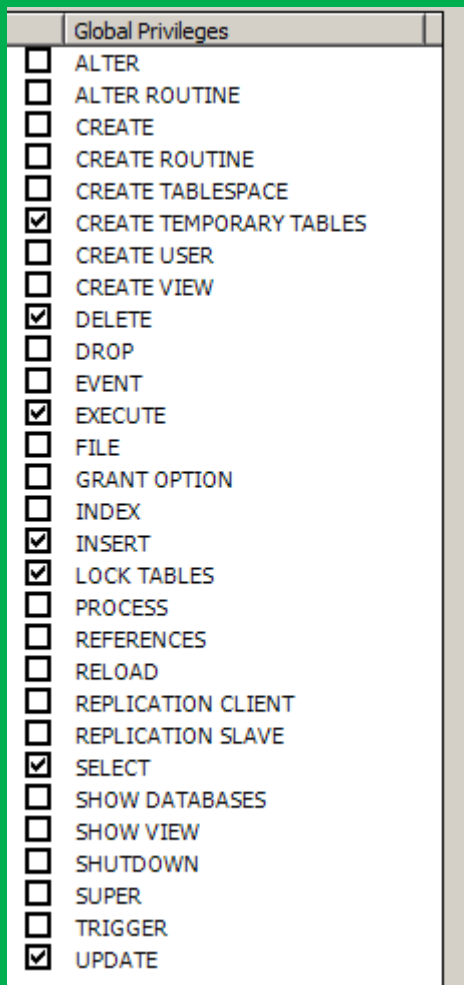
(Os "???" são para melhor ilustração, eles não estão assim no projeto normal)

Em paralelo a isso, decidi já estabelecer o meu banco de dados. Criei um usuário específico para o programa, para evitar que alguma espécie de bug pudesse afetar mais do que deveria o Banco de Dados, e criei as primeiras tabelas do projeto.



A screenshot of the MySQL User Creation Wizard. The 'Login Name' field contains 'BibliotecaM'. The 'Authentication Type' is set to 'Standard'. The 'Limit to Hosts Matching' field contains 'localhost'. The 'Password' and 'Confirm Password' fields are filled with asterisks. An 'Expire Password' button is at the bottom.

Login Name:	<input type="text" value="BibliotecaM"/>	You may create multiple accounts with the same name to connect from different hosts.
Authentication Type:	<input type="text" value="Standard"/>	For the standard password and/or host based authentication, select 'Standard'.
Limit to Hosts Matching:	<input type="text" value="localhost"/>	% and _ wildcards may be used
Password:	<input type="password" value="*****"/>	Type a password to reset it.
Confirm Password:	<input type="password" value="*****"/>	Enter password again to confirm.
<input type="button" value="Expire Password"/>		



A screenshot of the MySQL Global Privileges list. The list includes various privileges with checkboxes next to them. The checked privileges are: CREATE TEMPORARY TABLES, DELETE, EXECUTE, INSERT, LOCK TABLES, SELECT, and UPDATE.

Global Privileges	
<input type="checkbox"/>	ALTER
<input type="checkbox"/>	ALTER ROUTINE
<input type="checkbox"/>	CREATE
<input type="checkbox"/>	CREATE ROUTINE
<input type="checkbox"/>	CREATE TABLESPACE
<input checked="" type="checkbox"/>	CREATE TEMPORARY TABLES
<input type="checkbox"/>	CREATE USER
<input type="checkbox"/>	CREATE VIEW
<input checked="" type="checkbox"/>	DELETE
<input type="checkbox"/>	DROP
<input type="checkbox"/>	EVENT
<input checked="" type="checkbox"/>	EXECUTE
<input type="checkbox"/>	FILE
<input type="checkbox"/>	GRANT OPTION
<input type="checkbox"/>	INDEX
<input checked="" type="checkbox"/>	INSERT
<input checked="" type="checkbox"/>	LOCK TABLES
<input type="checkbox"/>	PROCESS
<input type="checkbox"/>	REFERENCES
<input type="checkbox"/>	RELOAD
<input type="checkbox"/>	REPLICATION CLIENT
<input type="checkbox"/>	REPLICATION SLAVE
<input checked="" type="checkbox"/>	SELECT
<input type="checkbox"/>	SHOW DATABASES
<input type="checkbox"/>	SHOW VIEW
<input type="checkbox"/>	SHUTDOWN
<input type="checkbox"/>	SUPER
<input type="checkbox"/>	TRIGGER
<input checked="" type="checkbox"/>	UPDATE

(Permissões recomendadas pelo instrutor Victorino Vila da Alura)

Na versão 0.2, criei uma classe para as categorias, uma inspiração do curso de JDBC, onde as categorias seriam uma tabela do Banco de Dados a parte, para padronizar sua escrita, sem ter o problema de um escrever a categoria com letras minúsculas e outro com maiúsculas, por exemplo.

Pós 17-09

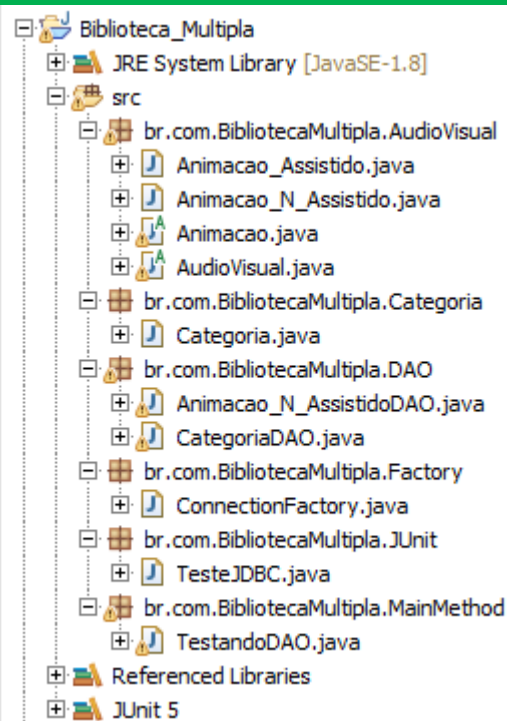
Versão 0.1

Recriação de todo o projeto, com base na utilização do JDBC; criação da classe "FactoryConnection".

Versão 0.2

Criação da classe "Categoria"; Criação e testagem da classe "CategoriaDAO".

(https://github.com/MatheusRO/BibliotecaMultipla/tree/main/Biblioteca_Multipla%2017-09%200.2)



```
package br.com.BibliotecaMultipla.DAO;

import java.sql.Connection;

public class CategoriaDAO {

    private Connection connection;

    public CategoriaDAO(Connection connection) {
        this.connection = connection;
    }

    public Collection<Categoria> listar() {
        try {
            Collection<Categoria> categorias = new ArrayList<>();
            String sql = "SELECT ID, NOME FROM CATEGORIA";

            try (PreparedStatement pstm = this.connection.prepareStatement(sql)) {
                pstm.execute();

                try (ResultSet rst = pstm.getResultSet()) {
                    while (rst.next()) {
                        Categoria categoria = new Categoria(rst.getInt(1), rst.getString(2));

                        categorias.add(categoria);
                    }
                }
            }
            return Collections.unmodifiableCollection(categorias);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

A versão 0.3, no que lhe concerne, teve uma quantia maior de atualizações. Criei o pacote “Pessoa”, que seriam as classes que representariam seres no projeto. E criei duas classes de pessoa: Dubladores e Personagens. Criei a tabela de Animação no Banco de dados, para ter a mesma hierarquia presente no projeto Java, além de criar uma classe exclusivamente para trabalhar com nome e ID. Isso se deve ao fato de eu perceber que toda a classe que representava uma tabela no banco necessariamente teria que ter ambos os atributos; para globalizar as restrições dessas informações, decidi criar uma classe que trabalharia com isso, além de uma “interface” para obrigar os métodos em envolvendo esses 2 atributos em todas as classes que os tivessem.

Pós 17-09

Versão 0.1

Recriação de todo o projeto, com base na utilização do JDBC; criação da classe "FactoryConnection".

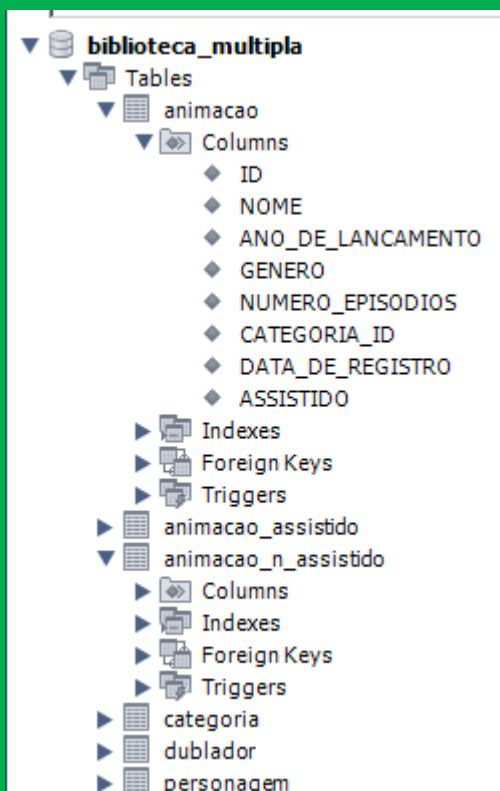
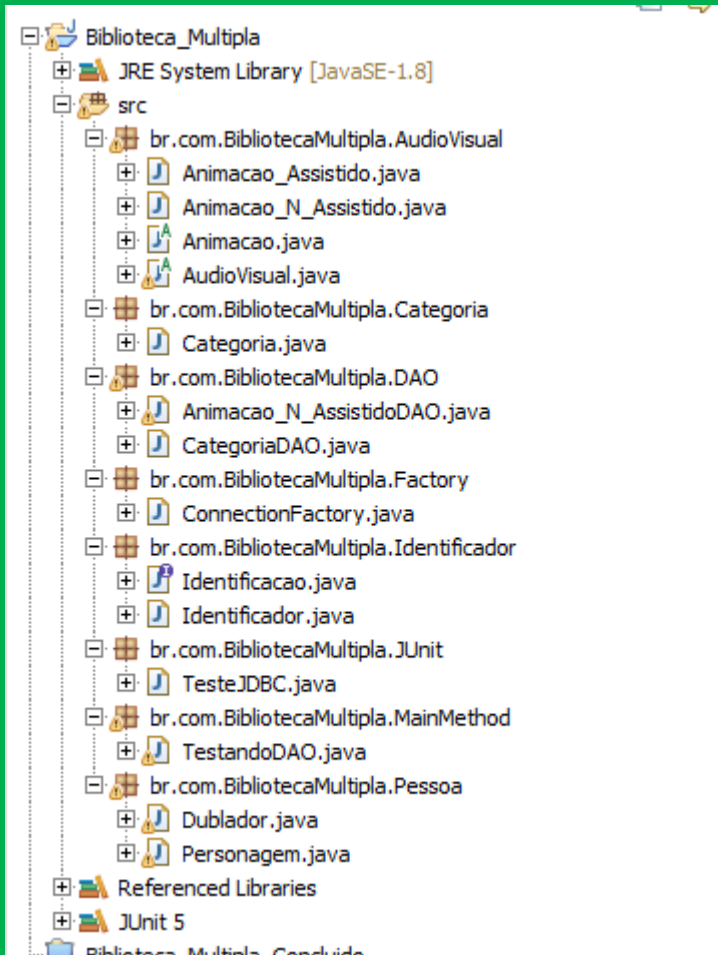
Versão 0.2

Criação da classe "Categoria"; Criação e testagem da classe "CategoriaDAO".

Versão 0.3

Criação do pacote "Pessoa"; criação da classe "Personagem"; criação da classe "Dublador"; atualização no banco de dados para ter a tabela mãe "Animacao" e as tabelas filhas "Animacao_N_Assistido" e "Animacao_Assistido"; criação do pacote "Identificador", com a interface "Identificacao" e a classe "Identificador", para englobar as regras de negocio referentes aos atributos "ID" e "Nome".

(https://github.com/MatheusRO/BibliotecaMultipla/tree/main/Biblioteca_Multipla%2017-09%200.3)



Na versão 0.4, eu criei DAOs para Personagem e Dublador, além de fazer o DAO pertencente as classes de animação serem apenas 1. Também criei uma tabela no Banco de Dados que trabalharia com as dublagens.

Por fim, eu um método para mudar uma animação de “não assistida” para “assistida”. Porém, para fazer isso, seria necessário mexer em 3 tabelas em simultâneo, no caso, de Animação para mudar o “boolean” de “assistido”, na tabela de “Não assistidas” para retirar a informação de lá e coloca-la na tabela de “assistidas”; para tal, eu precisaria de 3 conexões com o Banco de Dados. Após pensar muito, cheguei à conclusão que a melhor abordagem seria um “Stored Procedure” do próprio MySQL. Assim, eu poderia fazer uma chamada pelo Java, inserindo todas as informações necessárias, e o próprio Banco de Dados as redistribuiria para as classes pretendidas. Aproveitei e criei uma procedure para cada ação no Banco de Dados que trabalharia com mais de uma tabela.

🔗 Pós 17-09

Versão 0.1

Recriação de todo o projeto, com base na utilização do JDBC; criação da classe "FactoryConnection".

Versão 0.2

Criação da classe "Categoria"; Criação e testagem da classe "CategoriaDAO".

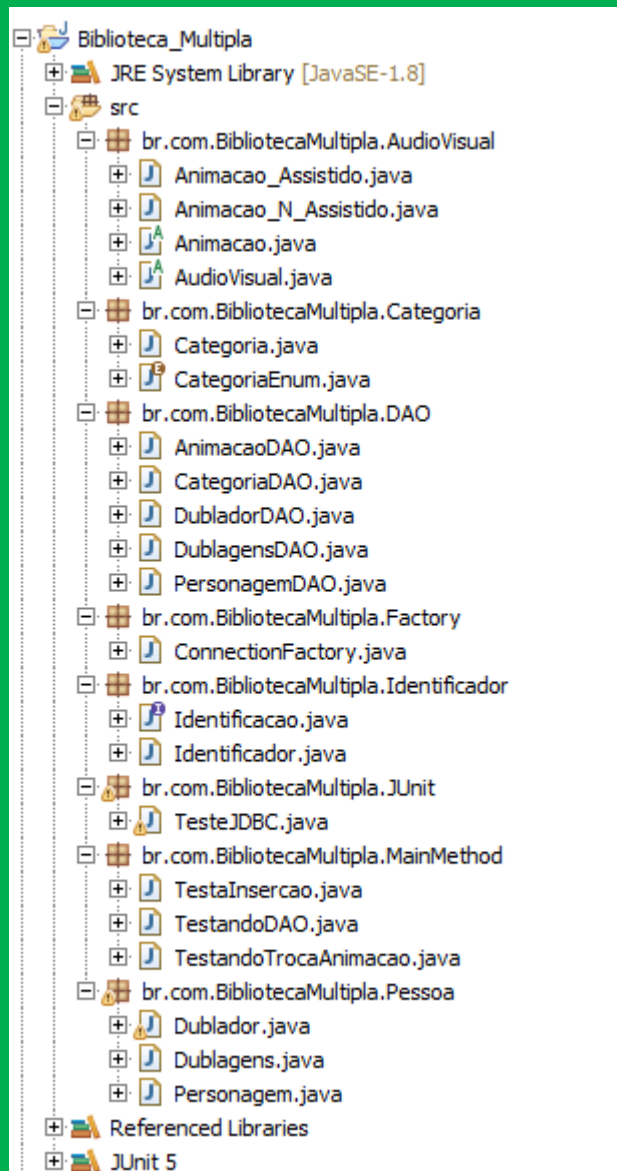
Versão 0.3

Criação do pacote "Pessoa"; criação da classe "Personagem"; criação da classe "Dublador"; atualização no banco de dados para ter a tabela mãe "Animacao" e as tabelas filhas "Animacao_N_Assistido" e "Animacao_Assistido"; criação do pacote "Identificador", com a interface "Identificacao" e a classe "Identificador", para englobar as regras de negocio referentes aos atributos "ID" e "Nome".

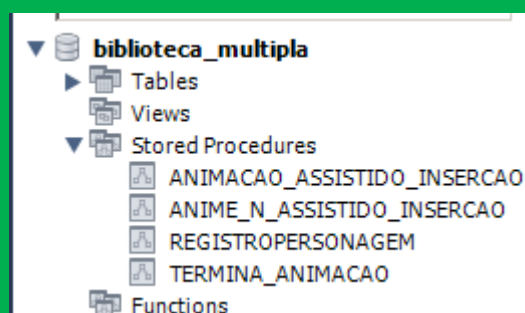
Versão 0.4

Criação das classes "DubladorDAO" e "PersonagemDAO"; Criação da tabela "Dublagens" no BD; Criação do método "FinalizarAnimação", que troca uma animação de "Não assistido" para "assistido", possibilitando a inserção de uma nota.

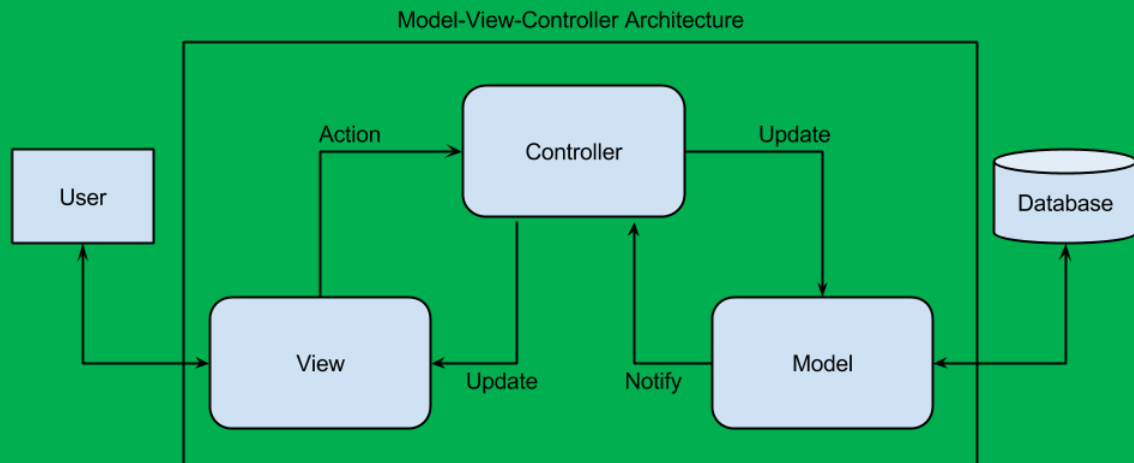
(https://github.com/MathheusRO/BibliotecaMultipla/tree/main/Biblioteca_Multipla%2019-09%200.4)



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `TERMINA_ANIMACAO`(cID INT, cNOTA DECIMAL (5,3))
BEGIN
    DELETE FROM ANIMACAO_N_ASSISTIDO WHERE ANIMACAO_ID = cID;
    INSERT INTO ANIMACAO_ASSISTIDO (ANIMACAO_ID, DATA_DE_FINALIZACAO, NOTA) VALUES (cID, DEFAULT, cNOTA);
    UPDATE ANIMACAO SET ASSISTIDO = 1 WHERE ID = cID;
END
```



Concluindo na versão 0.5, eu decidi me inspirar no modelo MVC (Model-View-Controller). Esse modelo é simples, uma classe vai trabalhar com o banco de dados (os DAO), uma classe vai dar comandos nessas classes de controle (Controllers), essas ordens serão enviadas pelo usuário a partir da tela de interação (View), e as informações obtidas pela classe DAO serão exibidas na tela de interação.



Então, eu criei as classes de controle, e criei um método main inicial, que simularia uma view, já que, no momento, eu não aprendi a trabalhar com algo nesse sentido.

Também modifiquei a estrutura do banco de dados, me desfazendo da tabela de “Dublagens”, deixando suas funções para a classe de personagem (medida tomada para a facilitação de troca de informação entre o banco e o JDBC).

Então, criei um JAR executável, para juntar tudo no projeto, e um arquivo DOC, para mostrar melhor o projeto para quem quer ver sem necessitar de ler o código.

Packages
Package
br.com.BibliotecaMultipla.AudioVisual
br.com.BibliotecaMultipla.Categoria
br.com.BibliotecaMultipla.Controller
br.com.BibliotecaMultipla.DAO
br.com.BibliotecaMultipla.Factory
br.com.BibliotecaMultipla.Identificador
br.com.BibliotecaMultipla.Inicializador
br.com.BibliotecaMultipla.Pessoa

Package br.com.BibliotecaMultipla.AudioVisual

Class Animacao

java.lang.Object

br.com.BibliotecaMultipla.AudioVisual.AudioVisual

br.com.BibliotecaMultipla.AudioVisual.Animacao

All Implemented Interfaces:

Identificacao

Direct Known Subclasses:

Animacao_Assistido, Animacao_N_Assistido

```
public class Animacao
extends AudioVisual
```

Representa o corpo de uma animacao

Version:

0.3

Author:

Matheus

```
Bem vindo a Biblioteca Multipla
Por favor, selecione a categoria com que quer interagir:
1 - Animacoes, 2 - Dubladores, 3 - Personagens
1
Escolha o tipo:
1 - Animacoes Assistidas; 2 - Animacoes nao assistidas; 3 - voltar
1
1 - Listar animacoes; 2 - Registrar animacao; 3 - voltar
2
Insira o nome do anime: Dragon Ball Z
Insira o ano de lancamento: 1989
Insira o(s) genero(s): Action, Adventure, Comedy, Fantasy
Insira a categoria que deseja:
1 - ANIMACAO_ORIENTAL
1 - ANIMACAO_OCIDENTAL
3 - SERIADO
4 - FILME
5 - JOGO_ELETRONICO
1
Insira o numero de episodios: 291
Insira sua nota entre 0 e 10 (Pode-se usar numeros quebrados, desde que separado
s por ?.?): 8.5
```

Animacao Dragon Ball Z registrada!
Fim do programa!

ID	NOME	ANO_DE_LANCAMENTO	GENERO	DATA_DE_REGISTRO	
1	He-Man and the Masters of the Universe	1983	Ação/aventura, fantasia científica	2021-09-30 11:03:45	A
2	Digimon Adventure	1999	Action, Adventure, Comedy, Fantasy	2021-09-30 11:03:46	A
5	Dragon Ball Z	1989	Action, Adventure, Comedy, Fantasy	2021-09-30 11:05:05	A

CATEGORIA	NUMERO_EPISODIOS	DATA_DE_FINALIZACAO	NOTA
ANIMACAO_OCIDENTAL	130	2021-09-30 11:03:46	8.500
ANIMACAO_ORIENTAL	54	2021-09-30 11:03:46	7.500
ANIMACAO_ORIENTAL	291	2021-09-30 11:05:05	8.500

Conclusões finais: O projeto foi algo simples, mas que me permitiu usar vários conhecimentos diferentes. Eu pude usar desde Enums (para trabalhar com as categorias), até Collections, Maps, JDBC, Administração de permissões o MySQL, criação de tabelas, buscas e até mesmo Stored Procedures.

Um agradecimento a Alura, que foi o pilar de tudo que eu aprendi aqui, e aos professores das formações que eu mencionei.

Links

Cursos:

<https://cursos.alura.com.br/formacao-oracle-mysql>

<https://cursos.alura.com.br/formacao-java>

<https://cursos.alura.com.br/course/jdbc-dao-persistencia>

Github:

<https://github.com/MatheusRQ/BibliotecaMultipla>

Linkedin:

<https://www.linkedin.com/in/matheus-rosa-de-queiroz-a76911205/>

Alura:

<https://cursos.alura.com.br/user/MatheusRQ>

Trello do projeto (as ideias são coisas que poderiam ser incluídas em uma atualização visando um projeto mais profissional):

The screenshot shows a Trello board with a background image of a starry night sky and mountains. The board is organized into five columns:

- Ideias**:
 - Criar classe "Ator"
 - Criar classes controllers para todas as classes DAO
 - Criar a classe filmes
 - Criar a classe Livros
 - Implementar e testar a interface Serializable na classe Animacao
 - + Adicionar um cartão
- Product Backlog (Tarefas)**:
 - + Adicionar um cartão
- A fazer (To do)**:
 - + Adicionar um cartão
- Projetos em andamento (Doing)**:
 - + Adicionar um cartão
- Concluídos (Done)**:
 - Sprint 1**: Criar o banco de dados do projeto
 - Sprint 1**: Criar Conexão com o BD (2/2)
 - Sprint 1**: Criar teste de conexão no JUnit
 - Sprint 2**: Criar a primeira Classe (Animacao) do projeto
 - Sprint 1**: Montar a primeira hierarquia de classes
 - Sprint 1**: Criar a "Animacao_N_AssistidoDAO"
 - Sprint 1**: Criar "CategoriaDAO"
 - Sprint 2**: Criar o teste de Unmodifiable na listas listas provenientes dos DAO (2/2)
 - Sprint 2**: Criar classe Dublador (2/2)
 - Criar classe Personagem (2/2)

Concluidos (Done) ...

Sprint 1

Modificar o nome da classe "Animacao_N_AssistidoDAO" para "AnimacaoDAO" e adicionar o método "listarAssistido"

Sprint 2

Adicionar o método "RegistrarNaoAssistido" na classe "AnimacaoDAO"

2/2

Sprint 1

Implementar método "FinalizarAnimacao" na classe "AnimacaoDAO"

Sprint 3

Mudar o nome da SP de "ANIME_N_ASSISTIDO_INSERTAO" para "ANIMACAO_N_ASSISTIDO_INSERTAO"

Sprint 1

Criar DubladorDAO e PersonagemDAO

2/2

Sprint 1

Criar tela inicial do programa

Sprint 1

Registrar todas as Categorias

Sprint 1

+ Adicionar um cartão

Sprint 1

Consertar o botão que devolve as dublagens de um dublador em específico.

Sprint 3

Descobrir o porquê de todas as animações assistidas retornarem categoria como "0" e consertar.

Sprint 2

Construir o JavaDoc do projeto

Sprint 1

Criar o documento de apresentação do projeto.