

# Programação Orientada a Objetos

## Interface Gráfica Persistência em Banco de Dados



Joinville Batista Junior

## Teste de Acesso ao BD

```
package controle;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class TesteBDLab {

    static final String URL_BD = "jdbc:mysql://localhost/bd_teste";
    static final String USUÁRIO = "root";
    static final String SENHA = "admin";
    private static Connection conexão = null;
    private static Statement comando = null;
```

## Teste de Acesso ao BD

```
public static void criaConexãoComando () {  
    try {  
        conexão = DriverManager.getConnection (URL_BD, USUÁRIO, SENHA);  
        comando = conexão.createStatement ();  
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}  
}  
  
public static void fechaComandoConexão () {  
    try {  
        comando.close();  
        conexão.close();  
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}  
}
```

## Teste de Acesso ao BD

```
public static void main(String[] args) {  
    criaConexãoComando ();  
    String sql;  
    sql = "DELETE FROM Pessoas";  
    try {  
        comando.executeUpdate(sql);  
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}  
    sql = "INSERT INTO Pessoas (CPF, Nome) VALUES ('111.111.111-11', 'Ana Julia')";  
    try {  
        comando.executeUpdate(sql);  
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}  
    sql = "SELECT Nome FROM Pessoas WHERE CPF = '111.111.111-11'";  
    ResultSet lista_resultados = null;  
    try {  
        lista_resultados = comando.executeQuery(sql);  
        while (lista_resultados.next()) {  
            System.out.println ("Nome: " + lista_resultados.getString("Nome"));  
        }  
        lista_resultados.close();  
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}  
    fechaComandoConexão ();  
}  
}
```

## Teste de Acesso ao BD

DROP TABLE IF EXISTS Pessoas;

CREATE TABLE Pessoas (  
CPF VARCHAR(15) NOT NULL PRIMARY KEY,  
Nome VARCHAR(50) NOT NULL);

## Passos do Teste

### 1 – Instale o BD

instale o SGBD MySQL

implante o MySQL Connector

    copie o arquivo 'mysql-connector-java-5.1.7-bin.jar' (ou  
    correspondente à versão mais atual)  
    para a pasta do jdk no diretório \jre\lib\ext

### 2 – Configure o BD na aba Banco de Dados da janela Serviços conecte o servidor MySQL

Host: localhost

Porta: 3306

Usuário: root

Senha: admin

crie o BD “bd\_teste” no Servidor MySQL

## Passos do Teste

### 3 – Crie a tabela Pessoas

no BD bd\_teste

selecione em Tabelas a opção Executar Comando

execute o script SQL de criação da tabela Pessoas

### 4 – Execute o teste de acesso ao BD

execute o exemplo de teste de acesso ao BD

o sistema

cria um registro na tabela Pessoas com nome e CPF

consulta o nome através do CPF

imprime o nome consultado

você deverá visualizar

Nome: Ana Julia

## Script SQL de Geração das Tabelas do BD

```
DROP TABLE IF EXISTS Clientes;  
DROP TABLE IF EXISTS Filmes;  
DROP TABLE IF EXISTS Reservas;
```

```
CREATE TABLE Clientes (  
    CPF VARCHAR(15) NOT NULL PRIMARY KEY,  
    Nome VARCHAR(50) NOT NULL,  
    Endereço VARCHAR(50) NOT NULL,  
    Telefone VARCHAR(12) NOT NULL );
```

```
CREATE TABLE Filmes (  
    Sequencial INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    Título VARCHAR(60) NOT NULL,  
    Assunto INT NOT NULL,  
    Categoria VARCHAR(15) NOT NULL,  
    Oscar BIT(1) NOT NULL );
```

```
CREATE TABLE Reservas (  
    Sequencial INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    Clienteld VARCHAR(15) NOT NULL,  
    Filmeld INT NOT NULL,  
    FOREIGN KEY (Clienteld) REFERENCES Clientes(CPF),  
    FOREIGN KEY (Filmeld) REFERENCES Filmes(Sequencial));
```

## Classe persistência.BD

```
package persistência;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class BD {

    static final String URL_BD = "jdbc:mysql://localhost/videolocadora";
    static final String USUÁRIO = "root";
    static final String SENHA = "admin";
    public static Connection conexão = null;
```

## Classe persistência.BD

```
public static void criaConexão () {
    try {
        conexão = DriverManager.getConnection (URL_BD, USUÁRIO, SENHA);
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}
}

public static void fechaConexão () {
    try {
        conexão.close();
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}
}
}
```

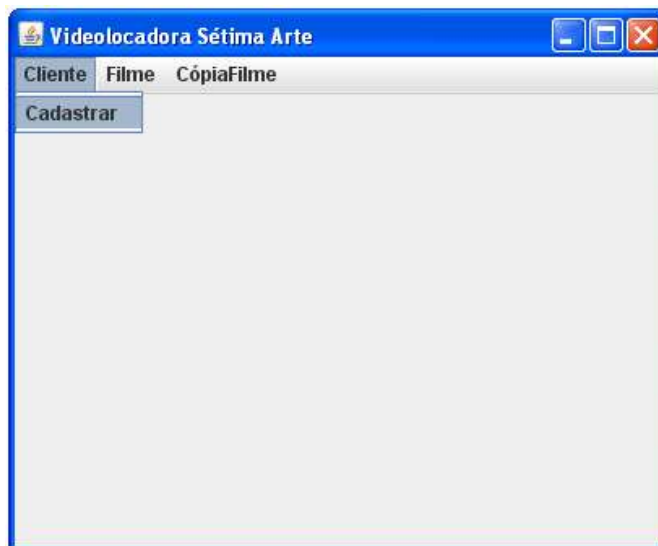
## Notas Gerais

- nesta apostila, os componentes gráficos serão referenciados nos textos explicativos, omitindo-se a letra J que caracteriza os componentes do pacote java.swing
  - desta forma um texto referenciando um componente Frame se refere ao componente swing JFrame, Label ao componente JLabel, e assim por diante
- por legibilidade as variáveis associadas aos componentes gráficos recebem um nome composto a partir do uso da componente e do tipo do componente, conforme os seguintes exemplos:
  - o componente JMenuItem utilizado para selecionar o Caso de Uso CadastrarCliente será denominado: cadastrar\_clienteMenuItem
  - o componente TextField utilizado para armazenar o nome do cliente será denominado: nomeTextField
- por simplicidade, o acesso ao BD está sendo suportado por métodos estáticos definidos nas classes de entidade
  - você pode optar por definir uma classe específica para a persistência dos dados das classes de entidades
    - como por exemplo, para a classe Cliente do pacote entidade
      - uma classe CClienteBD no pacote persistência
- propositamente, os Casos de Uso estão especificados com maior nível de detalhe, para facilitar a identificação da implementação

UFGD - POO-E4 - Joinville Batista Junior

11

## Janela do Sistema: Videolocadora Sétima Arte



UFGD - POO-E4 - Joinville Batista Junior

12

## Herança e Composição da Janela

### Herança

- a janela do sistema estende o componente Frame

### Composição: criar a seguinte hierarquia de componentes

- vincular, à janela, a barra de menus (MenuBar)
  - vincular, à barra de menu, os itens de menu (Menu), nomeando-os conforme a figura da Janela do Sistema
    - vincular, ao menu Cliente, os itens de menu (MenuItem):  
Cadastrar
    - vincular, ao menu Filme, os itens de menu: Cadastrar, Reservar
    - vincular, ao menu CópiaFilme, os itens de menu: Cadastrar, Locar, Devolver

## Nomes de Variáveis e Propriedades

### Nomes das Variáveis dos Componentes

- para todos os componentes utilizados, redefinir seu nome de variável conforme os nomes das variáveis do código da classe `interfaces.JanelaVideolocadora`

### Propriedades

- Frame
  - definir o título da janela (title)
  - definir operação de fechamento (defaultCloseOperation) como `DISPOSE`
  - definir tamanho preferido (preferredSize)

## Tratadores de Eventos e Inicialização

### Tratadores de Eventos

- associar, à janela, o tratador do evento `windowClosed`, denominado `terminarSistema`
  - encerrar o comando e a conexão com o BD
  - terminar a execução do sistema
- associar, a cada componente de item de menu, o correspondente tratador do evento `actionPerformed`
  - por exemplo, associar ao item de menu `CadastrarCliente` o tratador de evento `cadastrarCliente`
  - cada tratador de evento de um dado item de menu chama o controlador do respectivo Caso de Uso
  - os controladores de eventos, associados a Casos de Uso não implementados, ativam a visualização de uma janela secundária de diálogo informando: `Serviço Indisponível`

### Inicialização

- criar conexão com o BD e comando para execução de scripts SQL
  - por simplicidade, para um sistema local (sem acesso à web), foi criado um único comando para todos aos acessos ao BD

## Classe `interfaces.JanelaVideolocadora`

```
package interfaces;
```

```
import javax.swing.JOptionPane;  
import controle.ControladorCadastroCliente;  
import controle.ControladorCadastroFilme;  
import controle.ControladorReserva;  
import persistência.BD;
```

```
public class JanelaVideolocadora extends javax.swing.JFrame {
```

```
    public JanelaVideolocadora() {  
        BD.criaConexão();  
        initComponents();  
    }
```

```
    private void initComponents() { ... } // código gerado automaticamente
```



## Classe interfaces.JanelaVideolocadora

```
private void informaServiçoIndisponível () {
    JOptionPane.showMessageDialog (this, "Serviço Indisponível",
        "Informação", JOptionPane.INFORMATION_MESSAGE);
}

private void terminarSistema(java.awt.event.WindowEvent evt) {
    BD.fechaConexão();
    System.exit(1);
}

private void cadastrarCliente(java.awt.event.ActionEvent evt) {
    new ControladorCadastroCliente ();
}

private void cadastrarFilme(java.awt.event.ActionEvent evt) {
    new ControladorCadastroFilme ();
}

private void reservarFilme(java.awt.event.ActionEvent evt) {
    new ControladorReserva();
}
```

## Classe interfaces.JanelaVideolocadora

```
private void cadastrarCópiaFilme(java.awt.event.ActionEvent evt) {
    informaServiçoIndisponível ();
}

private void locarFilme(java.awt.event.ActionEvent evt) {
    informaServiçoIndisponível ();
}

private void devolverFilme(java.awt.event.ActionEvent evt) {
    informaServiçoIndisponível ();
}

public static void main(String args[]) { // código gerado automaticamente
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new JanelaVideolocadora().setVisible(true);
        }
    });
}
```

## Classe interfaces.JanelaVideolocadora

```
// Variables declaration - do not modify
private javax.swing.JMenuItem cadastrar_clienteMenuItem;
private javax.swing.JMenuItem cadastrar_cópiaMenuItem;
private javax.swing.JMenu clienteMenu;
private javax.swing.JMenu cópia_filmeMenu;
private javax.swing.JMenuItem devolverMenuItem;
private javax.swing.JMenu filmeMenu;
private javax.swing.JMenuItem locarMenuItem;
private javax.swing.JMenuItem reservar_filmeMenuItem;
private javax.swing.JMenuBar videolocadoraMenuBar;
// End of variables declaration
```

## Janela do Caso de Uso: Cadastrar Clientes



The screenshot shows a Java Swing window titled "Cadastrar Clientes". Inside the window, there is a label "Clientes Cadastrados" followed by a dropdown arrow. Below this, there are four text input fields labeled "CPF", "Nome", "Endereço", and "Telefone". At the bottom of the window, there are five buttons: "Inserir", "Consultar", "Alterar", "Remover", and "Limpar".

## Herança e Composição da Janela

### Herança

- a janela do sistema estende o componente Frame

### Composição: vincular à janela os seguintes componentes

- Label e ComboBox : para visualização e seleção dos clientes cadastrados
- Label e TextField: 4 pares
  - para preenchimento ou visualização dos campos do cadastro do cliente: CPF, nome, endereço e telefone
- Panel: painel de comandos do Caso de Uso
  - manter o layout padrão do container Panel: FlowLayout
  - vincular ao painel, os botões (Button) associados aos comandos do Caso de Uso

## Formatação do Layout

### Formatação do Layout

- alterar o layout padrão (BorderLayout) do Frame para GridBagLayout
- posicionar os componentes nas células da grade
- associar o painel de comando à ocupação de duas colunas
- alinhar os labels à direita e os seus componentes associados à esquerda
- distanciar as linhas e colunas de componentes da grade

## Nomes de Variáveis e Propriedades

### Nomes das Variáveis dos Componentes

- para todos os componentes utilizados, redefinir seu nome de variável conforme os nomes das variáveis do código da classe `interfaces.JanelaCadastroCliente`

### Ajustar propriedades dos componentes

- Labels: ajustar o texto de seus rótulos (text)
- TextFields
  - remover seus textos internos (text)
  - ajustar o tamanho dos campos de textos (columns), em função da informação associada a cada campo
- ComboBox
  - associar o conjunto de visões de clientes como modelo (model)
    - `setModel (new DefaultComboBoxModel (clientes_cadastrados))`
- Buttons: ajustar seus textos (text)

## Tratadores de Eventos

### Tratadores de eventos

- associar, a cada um dos botões de comando, o correspondente tratador do evento `actionPerformed`

### Tratador do evento gerado no botão Inserir

- se todos os campos tiverem sido preenchidos
  - cria objeto cliente a partir dos campos de texto informados pelo usuário
- solicita a execução da ação ao controlador
  - regra negócio: não deve existir cliente cadastrado com a chave (CPF) do cliente a ser inserido
  - solicita ao BD
    - cliente cadastrado a partir da chave: para checar regra de negócio
    - inserção de registro de cliente
- extrai a visão do cliente, adiciona ao ComboBox de clientes cadastrados e assinala como último elemento selecionado
- em caso de insucesso (algum campo não preenchido ou falha reportada pelo controlador): informa mensagem de erro

## Tratadores de Eventos

Tratador do evento gerado no botão Consultar

- obtém a visão selecionada no ComboBox de clientes cadastrados
- busca no BD, o cliente com a chave associada à visão selecionada
  - não existe nenhuma regra de negócios a ser checada pelo controlador
- preenche os campos de texto do cadastro com os atributos do objeto cliente
- em caso de insucesso (inconsistência devido a cliente não cadastrado com a chave selecionada): informa mensagem de erro

## Tratadores de Eventos

Tratador do evento gerado no botão Alterar

- se todos os campos tiverem sido preenchidos
  - cria objeto cliente a partir dos campos de texto informados pelo usuário
- solicita a execução da ação ao controlador
  - regra negócio: deve existir um cliente cadastrado com a chave do cliente a ser alterado
  - solicita ao BD
    - cliente cadastrado a partir da chave: para checar regra de negócio
    - alteração do registro de cliente
- obtém o objeto visão dos clientes cadastrados (a partir da chave do cliente), altera a informação da visão considerando a visão obtida a partir do cliente alterado, atualiza o ComboBox de clientes cadastrados e seleciona item alterado
- em caso de insucesso (algum campo não preenchido ou falha reportada pelo controlador): informa mensagem de erro

Nota

- é recomendável que o usuário consulte previamente o cliente que deseja alterar, para alterar os atributos desejados, com exceção da chave

## Tratadores de Eventos

Tratador do evento gerado no botão Remover

- obtém a visão selecionada no ComboBox de clientes cadastrados
- solicita a execução da ação ao controlador
  - regra negócio: deve existir um cliente cadastrado com a chave do cliente a ser removido
  - solicita ao BD
    - cliente cadastrado a partir da chave: para checar regra de negócio
    - a remoção do registro do cliente
- remove a visão selecionada no ComboBox de clientes cadastrados e seleciona o primeiro item da lista
  - ou nenhum item, se o último item foi removido
- em caso de insucesso (nenhum cliente selecionado para remoção ou falha reportada pelo controlador): informa mensagem de erro

Tratador do evento gerado no botão Limpar

- remove o conteúdo dos campos de texto do cadastro do cliente

## Inicialização

Inicialização

- armazenar o objeto do controlador do Caso de Uso para futuras chamadas de funcionalidades associadas a eventos gerados pelo usuário
- a partir do BD, atualizar conjunto de visões associados ao componente ComboBox, utilizado para selecionar clientes cadastrados

## Classe controle.ControladorCadastroCliente

```
package controle;

import entidade.Cliente;
import interfaces.JanelaCadastroCliente;

public class ControladorCadastroCliente {

    public ControladorCadastroCliente() {
        new JanelaCadastroCliente(this).setVisible(true);
    }

    public String inserirCliente (Cliente cliente) {
        Cliente cliente1 = Cliente.buscarCliente (cliente.getCpf ());
        if (cliente1 == null) {
            return Cliente.inserirCliente (cliente);
        } else {
            return "CPF de cliente já cadastrado";
        }
    }
}
```

## Classe controle.ControladorCadastroCliente

```
public String alterarCliente (Cliente cliente) {
    Cliente cliente1 = Cliente.buscarCliente (cliente.getCpf());
    if (cliente1 != null) {
        return Cliente.alterarCliente (cliente);
    } else {
        return "CPF de cliente não cadastrado";
    }
}

public String removerCliente (String cpf) {
    Cliente cliente1 = Cliente.buscarCliente (cpf);
    if (cliente1 != null) {
        return Cliente.removerCliente (cpf);
    } else {
        return "CPF de cliente não cadastrado";
    }
}
}
```

## Classe interfaces.JanelaCadastroCliente

```
package interfaces;

import javax.swing.JOptionPane;
import controle.ControladorCadastroCliente;
import entidade.Cliente;
import entidade.Visão;
import java.util.Vector;
import javax.swing.DefaultComboBoxModel;

public class JanelaCadastroCliente extends javax.swing.JFrame {

    ControladorCadastroCliente controlador;
    Vector<Visão<String>> clientes_cadastrados;

    public JanelaCadastroCliente(ControladorCadastroCliente controlador) {
        this.controlador = controlador;
        clientes_cadastrados = Cliente.getVisões();
        initComponents();
    }

    private void initComponents() { ... } // código gerado automaticamente
```

## Classe interfaces.JanelaCadastroCliente

```
private void inserirCliente(java.awt.event.ActionEvent evt) {
    Cliente cliente = obterClienteInformado();
    String mensagem_erro = null;
    if (cliente != null) mensagem_erro = controlador.inserirCliente(cliente);
    else mensagem_erro = "Algum atributo do cliente não foi informado";
    if (mensagem_erro == null) {
        Visão<String> visão = cliente.getVisão();
        clientes_cadastradosComboBox.addItem(visão);
        clientes_cadastradosComboBox.setSelectedItem(visão);
    } else JOptionPane.showMessageDialog(this, mensagem_erro, "ERRO",
        JOptionPane.ERROR_MESSAGE);
}
```



## Classe interfaces.JanelaCadastroCliente

```
private void consultarCliente(java.awt.event.ActionEvent evt) {
    Visão<String> visão
        = (Visão<String>) clientes_cadastradosComboBox.getSelectedItem ();
    Cliente cliente = null;
    String mensagem_erro = null;
    if (visão != null) {
        cliente = Cliente.buscarCliente (visão.getChave());
        if (cliente == null) mensagem_erro = "Cliente não cadastrado";
    } else mensagem_erro = "Nenhum cliente selecionado";
    if (mensagem_erro == null) {
        cpfTextField.setText (cliente.getCpf());
        nomeTextField.setText (cliente.getNome());
        endereçoTextField.setText (cliente.getEndereço());
        telefoneTextField.setText (cliente.getTelefone());
    } else JOptionPane.showMessageDialog
        (this, mensagem_erro, "ERRO", JOptionPane.ERROR_MESSAGE);
}
```

## Classe interfaces.JanelaCadastroCliente

```
private void alterarCliente(java.awt.event.ActionEvent evt) {
    Cliente cliente = obterClienteInformado();
    String mensagem_erro = null;
    if (cliente != null) mensagem_erro = controlador.alterarCliente(cliente);
    else mensagem_erro = "Algum atributo do cliente não foi informado";
    if (mensagem_erro == null) {
        Visão<String> visão = getVisãoClientesCadastrados(cliente.getCpf());
        if (visão != null) {
            visão.setInfo(cliente.getVisão().getInfo());
            clientes_cadastradosComboBox.updateUI();
            clientes_cadastradosComboBox.setSelectedItem(visão);
        }
    } else JOptionPane.showMessageDialog
        (this, mensagem_erro, "ERRO", JOptionPane.ERROR_MESSAGE);
}
```

## Classe interfaces.JanelaCadastroCliente

```
private void removerCliente(java.awt.event.ActionEvent evt) {
    Visão<String> visão
        = (Visão<String>) clientes_cadastradosComboBox.getSelectedItem ();
    String mensagem_erro = null;
    if (visão != null) mensagem_erro = controlador.removerCliente(visão.getChave());
    else mensagem_erro = "Nenhum cliente selecionado";
    if (mensagem_erro == null) {
        clientes_cadastrados.remove(visão);
        if (clientes_cadastrados.size() >= 1)
            clientes_cadastradosComboBox.setSelectedIndex(0);
        else clientes_cadastradosComboBox.setSelectedIndex(-1);
    } else {
        JOptionPane.showMessageDialog(this, mensagem_erro, "ERRO",
            JOptionPane.ERROR_MESSAGE);
    }
}
```

## Classe interfaces.JanelaCadastroCliente

```
private Cliente obterClienteInformado() {
    String cpf = cpfTextField.getText();
    if (cpf.isEmpty()) return null;
    String nome = nomeTextField.getText();
    if (nome.isEmpty()) return null;
    String endereço = endereçoTextField.getText();
    if (endereço.isEmpty()) return null;
    String telefone = telefoneTextField.getText();
    if (telefone.isEmpty()) return null;
    return new Cliente (cpf, nome, endereço, telefone);
}

private void limparCamposTexto(java.awt.event.ActionEvent evt) {
    cpfTextField.setText("");
    nomeTextField.setText("");
    endereçoTextField.setText("");
    telefoneTextField.setText("");
}
```

## Classe interfaces.JanelaCadastroCliente

```
private Visão<String> getVisãoClientesCadastrados(String chave) {  
    for (Visão<String> visão : clientes_cadastrados) {  
        if (visão.getChave().equals(chave)) return visão;  
    }  
    return null;  
}
```

## Classe interfaces.JanelaCadastroCliente

```
// Variables declaration - do not modify  
private javax.swing.JButton alterarButton;  
private javax.swing.JComboBox clientes_cadastradosComboBox;  
private javax.swing.JLabel clientes_cadastradosLabel;  
private javax.swing.JPanel comandosPanel;  
private javax.swing.JButton consultarButton;  
private javax.swing.JLabel cpfLabel;  
private javax.swing.JTextField cpfTextField;  
private javax.swing.JLabel endereçoLabel;  
private javax.swing.JTextField endereçoTextField;  
private javax.swing.JButton inserirButton;  
private javax.swing.JButton limparButton;  
private javax.swing.JLabel nomeLabel;  
private javax.swing.JTextField nomeTextField;  
private javax.swing.JButton removerButton;  
private javax.swing.JLabel telefoneLabel;  
private javax.swing.JTextField telefoneTextField;  
// End of variables declaration
```

## Classe entidade.Visão

```
package entidade;

public class Visão<T> {

    T chave;
    String info;

    public Visão(T chave, String info) {
        this.chave = chave;
        this.info = info;
    }

    public T getChave() {
        return chave;
    }

    public void setChave(T chave) {
        this.chave = chave;
    }
}
```

## Classe entidade.Visão

```
public String getInfo() {
    return info;
}

public void setInfo(String info) {
    this.info = info;
}

@Override
public String toString () { return info; }
}
```

## Classe entidade.Cliente

```
package entidade;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Vector;
import persistência.BD;

public class Cliente {

    public static String removerCliente (String cpf) {
        String sql = "DELETE FROM Clientes WHERE CPF = ?";
        try {
            PreparedStatement comando = BD.conexão.prepareStatement(sql);
            comando.setString(1, cpf);
            comando.executeUpdate();
            comando.close();
            return null;
        } catch (SQLException exceção_sql) {
            exceção_sql.printStackTrace ();
            return "Erro na Remoção do Cliente no BD";
        }
    }
}
```

## Classe entidade.Cliente

```
public static Cliente buscarCliente (String cpf) {
    String sql = "SELECT Nome, Endereço, Telefone FROM Clientes"
        + " WHERE CPF = ?";
    ResultSet lista_resultados = null;
    Cliente cliente = null;
    try {
        PreparedStatement comando = BD.conexão.prepareStatement(sql);
        comando.setString(1, cpf);
        lista_resultados = comando.executeQuery();
        while (lista_resultados.next()) {
            cliente = new Cliente (cpf,
                lista_resultados.getString("Nome"),
                lista_resultados.getString("Endereço"),
                lista_resultados.getString("Telefone"));
        }
        lista_resultados.close();
        comando.close();
    } catch (SQLException exceção_sql) {
        exceção_sql.printStackTrace ();
        cliente = null;
    }
    return cliente;
}
```

### Classe entidade.Cliente

```
public static String inserirCliente (Cliente cliente) {
    String sql = "INSERT INTO Clientes (CPF, Nome, Endereço, Telefone)"
        + " VALUES (?, ?, ?, ?)";

    try {
        PreparedStatement comando = BD.conexão.prepareStatement(sql);
        comando.setString(1, cliente.getCpf());
        comando.setString(2, cliente.getNome());
        comando.setString(3, cliente.getEndereço());
        comando.setString(4, cliente.getTelefone());
        comando.executeUpdate();
        comando.close();
        return null;
    } catch (SQLException exceção_sql) {
        exceção_sql.printStackTrace ();
        return "Erro na Inserção do Cliente no BD";
    }
}
```

### Classe entidade.Cliente

```
public static String alterarCliente (Cliente cliente) {
    String sql = "UPDATE Clientes SET Nome = ?, Endereço = ?, Telefone = ?"
        + " WHERE CPF = ?";

    try {
        PreparedStatement comando = BD.conexão.prepareStatement(sql);
        comando.setString(1, cliente.getNome());
        comando.setString(2, cliente.getEndereço());
        comando.setString(3, cliente.getTelefone());
        comando.setString(4, cliente.getCpf());
        comando.executeUpdate();
        comando.close();
        return null;
    } catch (SQLException exceção_sql) {
        exceção_sql.printStackTrace ();
        return "Erro na Alteração do Cliente no BD";
    }
}
```

## Classe entidade.Cliente

```
public static Vector<Visão<String>> getVisões () {
    String sql = "SELECT CPF, Nome FROM Clientes";
    ResultSet lista_resultados = null;
    Vector<Visão<String>> visões = new Vector<Visão<String>> ();
    String cpf;
    try {
        PreparedStatement comando = BD.conexão.prepareStatement(sql);
        lista_resultados = comando.executeQuery();
        while (lista_resultados.next()) {
            cpf = lista_resultados.getString("CPF");
            visões.addElement(new Visão<String> (cpf,
                lista_resultados.getString("Nome") + " - " + cpf));
        }
        lista_resultados.close();
        comando.close();
    } catch (SQLException exceção_sql) {exceção_sql.printStackTrace ();}
    return visões;
}
```

## Classe entidade.Cliente

```
private String cpf, nome, endereço, telefone;

public Cliente(String cpf, String nome, String endereço, String telefone) {
    this.cpf = cpf;
    this.nome = nome;
    this.endereço = endereço;
    this.telefone = telefone;
}

public Cliente(String nome, String telefone) {
    this.nome = nome;
    this.telefone = telefone;
}

public Visão<String> getVisão () {
    return new Visão<String> (cpf, nome + " - " + cpf);
}

public String getCpf() {
    return cpf;
}
```

## Classe entidade.Cliente

```
public void setCpf(String cpf) {  
    this.cpf = cpf;  
}  
  
public String getEndereço() {  
    return endereço;  
}  
  
public void setEndereço(String endereço) {  
    this.endereço = endereço;  
}  
  
public String getNome() {  
    return nome;  
}  
  
public void setNome(String nome) {  
    this.nome = nome;  
}
```

## Classe entidade.Cliente

```
public String getTelefone() {  
    return telefone;  
}  
  
public void setTelefone(String telefone) {  
    this.telefone = telefone;  
}  
  
@Override  
public String toString () {  
    return nome + " - " + cpf;  
}  
}
```



## Exercícios

1 – Implemente o Caso de Uso Cadastrar Filmes com as seguintes funcionalidades:

- inserir um novo filme
  - o usuário deverá preencher o formulário de cadastro do filme (título, assunto, categoria [catálogo, lançamento, promoção], oscar de melhor filme [sim, não]) e solicitar sua criação no BD
  - se algum dado de cadastro filme não foi informado pelo usuário, se já existir um filme com todos os atributos coincidentes ou se não for possível cadastrar o novo filme no BD: o sistema deverá informar uma mensagem de erro e abortar a operação
  - após a criação do cadastro do filme no BD, o sistema deverá informar o identificador sequencial, atribuído ao filme pelo BD, e atualizar a lista de filmes cadastrados, informando o identificador sequencial e o título do filme criado

## Exercícios

- consultar um filme da lista de filmes cadastrados
  - o usuário deverá selecionar um filme, de uma lista de filmes cadastrados, e consultar seus dados no BD
  - se o filme selecionado não existir no BD (o que caracterizará erro de implementação) ou não for possível consultar os dados do filme no BD: o sistema deverá informar uma mensagem de erro e abortar a operação
  - o sistema deverá informar os atributos do filme consultado

## Exercícios

- alterar dados de um filme cadastrado
  - o usuário deverá consultar um filme cadastrado, alterar os atributos que desejar (com exceção do identificador sequencial, não habilitado para alteração) e solicitar sua alteração na BD
  - se o filme selecionado não existir no BD (o que caracterizará erro de implementação) ou não for possível alterar o cadastro do filme no BD: o sistema deverá informar uma mensagem de erro e abortar a operação
  - após a alteração do cadastro do filme no BD, o sistema deverá atualizar a lista de filmes cadastrados, caso o título do filme tenha sido alterado

## Exercícios

- remover um filme cadastrado
- o usuário deverá selecionar um filme, de uma lista de filmes cadastrados, e solicitar sua remoção do BD
- se o filme selecionado não existir no BD (o que caracterizará erro de implementação) ou não for possível remover o filme no BD: o sistema deverá informar uma mensagem de erro e abortar a operação
  - após a remoção do cadastro do filme, o sistema deverá atualizar a lista de filmes cadastrados, excluindo as informações associadas ao filme removido

Implemente, também, a funcionalidade para limpar os campos do cadastro

- removendo a também a seleção do ComboBox de assuntos e dos grupos de botões de rádio de categoria

## Exercícios

### 1.1 – Crie a classe entidade.Filme:

- tipo enumerado Assunto [aventura, comédia, drama, faroeste, ficção, guerra, romance, suspense, terror ]
- dados de objeto
  - dados: identificador sequencial, título, assunto, categoria [catálogo, lançamento, promoção], oscar de melhor filme (sim ou não)
- métodos de objeto
  - construtor
  - leitura e escrita dos atributos do objeto
  - Visão<Integer> getVisão ()
- dados da classe (estáticos)
  - array de assuntos: para conversão do índice no enumerado (armazenado no BD) no valor do enumerado (armazenado no objeto)
  - OBS: para a converter o valor no índice, aplique ordinal()

## Exercícios

- métodos da classe (estáticos)
  - Filme buscarFilme (int sequencial)
    - usar array de assuntos para converter índice do BD no enumerado
  - boolean existeFilmeMesmosAtributos (Filme filme)
    - aplicar ordinal() a valor enumerado para obter índice do BD
  - Vector<Visão<Integer>> getVisões ()
  - String inserirFilme (Filme filme)
    - idem aplicar ordinal()
  - int últimoSequencial ()
    - SQL : SELECT MAX(Sequencial)
  - String alterarFilme (Filme filme)
    - idem aplicar ordinal()
  - String removerFilme (int sequencial)

## Exercícios

1.2 – Crie a classe controle.ControladorCadastroFilme, atendendo às regras de negócio especificadas no Caso de Uso:

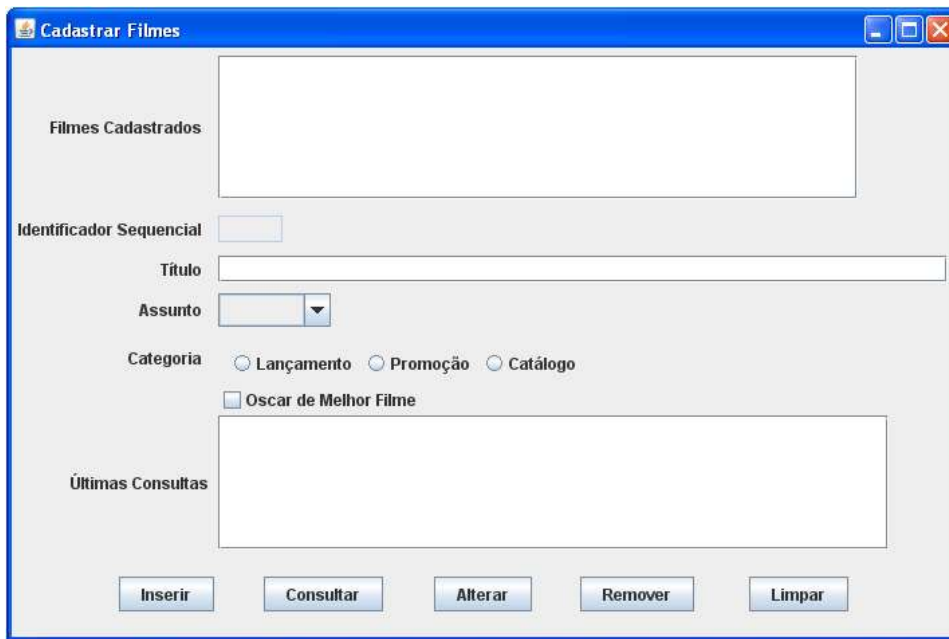
- ControladorCadastroFilme()
- String inserirFilme (Filme filme)
- String alterarFilme (Filme filme)
- String removerFilme (int sequencial)

## Exercícios

1.3 – Crie a classe interfaces.JanelaCadastroFilme, conforme a figura do slide seguinte:

- dados adicionais às variáveis do componentes da interface gráfica
  - ControladorCadastroFilme controlador
  - DefaultListModel modelo\_lista\_filmes

## Exercícios



UFGD - POO-E4 - Joinville Batista Junior

57

## Exercícios

### 1.3.1 – Composição

- labels dos componentes: Label
- filmes cadastrados: ScrollPane encapsulando List
- identificador sequencial e título: TextField
- assunto: ComboBox
- categoria: Panel com 3 RadioButton, e ButtonGroup para agrupar os botões de rádio
- oscar: CheckBox
- últimas consultas: ScrollPane encapsulando TextArea
- painel de comandos do Caso de Uso: Panel
  - vincular ao painel, os botões (Button) associados aos comandos do Caso de Uso

UFGD - POO-E4 - Joinville Batista Junior

58

## Exercícios

### 1.3.2 – Propriedades específicas

- List dos filmes cadastrados
  - associar modelo: `setModel (new DefaultListModel())`
  - definir modo de seleção (`selectionMode`) para único item (`SINGLE`)
- TextField do identificador sequencial
  - marcar como não editável (`editable`): sequencial é definido pelo BD e informado pelo sistema
- ComboBox do assunto (valores fixos)
  - associar modelo a array de valores do enumerado `Filme.Assuntos`
    - `new DefaultComboBoxModel (Filme.ASSUNTOS)`
- cada `RadioButton` da categoria
  - definir texto (`text`)
  - vincular ao grupo de botões (`categoriaButtonGroup`)
- `TextArea` das últimas consultas
  - definir número de linhas (`rows`) e colunas (`columns`)

## Exercícios

### 1.3.3 – Inicialização

- `inicializarListaFilmes ()`
  - obter modelo do List de objetos cadastrados (`getModel()`) e salvar
  - obter visões de filmes do BD
  - iterar nas visões para armazená-las no modelo do List de objetos cadastrados
- `inicializarCategoriasFilmes ()`
  - associar string a cada botão de radio (`setActionCommand (String)`)

## Exercícios

### 1.3.4 – Tratadores de Eventos

- void inserirFilme (ActionEvent)
  - criar objeto filme com sequencial nulo (provisório)
    - assunto: item selecionado do ComboBox de assuntos
    - categoria selecionada do grupo de botões de rádio
      - aplicar `getSelection().getActionCommand()` ao grupo de botões
    - oscar: aplicar `isSelected()` ao CheckBox
  - solicitar criação do filme ao controlador
  - após criar o filme no BD
    - ler último sequencial criado pelo BD (só funciona para um sistema não concorrente) e atualizar sequencial do objeto filme
    - inserir visão do filme no modelo do List de filmes cadastrados
    - posicionar lista de filmes no último elemento criado
    - mostrar o sequencial do filme criado

UFGD - POO-E4 - Joinville Batista Junior

61

## Exercícios

- void consultarFilme (ActionEvent)
  - obter visão de filme selecionada do List: `getSelectedValue()`
  - obter chave da visão e consultar o filme no BD
  - atualizar os componentes a partir dos atributos do filme consultado
    - marcaCategoriaFilme()
      - compara a categoria com o string armazenado previamente em cada botão de radio (`getActionCommand()`)
        - » seleciona o botão de rádio cuja categoria coincidir
    - apendarConsultaAreaTexto(Filme)
      - montar string do filme selecionado e apendar na área de texto das últimas consultas
        - » `append(String)`

UFGD - POO-E4 - Joinville Batista Junior

62

## Exercícios

- void alterarFilme (ActionEvent)
  - criar objeto filme com sequencial já existente
    - idem à inserção de filme para os demais atributos
  - solicitar a alteração do filme ao controlador
  - após alterar o filme no BD
    - obtém o objeto visão dos filmes cadastrados (a partir da chave do filme), altera a informação da visão considerando a visão obtida a partir do filme alterado, atualiza o List de filmes cadastrados

## Exercícios

- void removerFilme (ActionEvent)
  - obtém a visão selecionada no List de filmes cadastrados
  - solicita a remoção ao controlador
  - remove a visão selecionada do modelo do List de filmes cadastrados



## Exercícios

### 2 – Implemente o Caso de Uso Reservar Filmes:

- inserir uma nova reserva
  - o usuário deverá selecionar o cliente e o filme que caracterizam a nova reserva
  - se o cliente ou filme não foram informados pelo usuário, se já existir uma reserva para o par cliente e filme, ou se não for possível cadastrar a nova reserva no BD: o sistema deverá informar uma mensagem de erro e abortar a operação
  - após a criação do cadastro da reserva no BD, o sistema deverá informar o identificador sequencial, atribuído à reserva pelo BD, e limpar as informações que não estão relacionadas com a reserva criada

## Exercícios

- pesquisar reservas cadastradas
  - o usuário deverá selecionar o cliente e o filme utilizar os seguintes filtros para pesquisar reservas: cliente, filme e assunto do filme
  - o sistema deverá informar, na área de reservas pesquisadas, todas as reservas que satisfaçam os filtros selecionados

## Exercícios

- remover uma reserva cadastrada
  - o usuário deverá selecionar o cliente e o filme que caracterizam a reserva a ser removida
  - se o cliente ou filme não foram informados pelo usuário, se não existir uma reserva para o par cliente e filme, ou se não for possível remover a reserva no BD: o sistema deverá informar uma mensagem de erro e abortar a operação

Implemente, também, a funcionalidade para limpar os campos do cadastro

- removendo todas as seleções realizadas

## Exercícios

2.1 – Crie a classe entidade.Reserva:

- dados de objeto
  - dados: identificador sequencial, cliente, filme
- métodos de objeto
  - construtor
  - leitura e escrita dos atributos do objeto
- métodos da classe (estáticos)
  - int últimoSequencial ()
  - boolean existeReserva(String chave\_cliente, int chave\_filme)
  - String inserirReserva (String chave\_cliente, int chave\_filme)
  - Vector<Reserva> pesquisarReservas  
(String chave\_cliente, int chave\_filme, int assunto\_filme)
  - String removerReserva (String chave\_cliente, int chave\_filme)

## Exercícios

2.2 – Crie a classe controle.ControladorReserva , atendendo às regras de negócio especificadas no Caso de Uso :

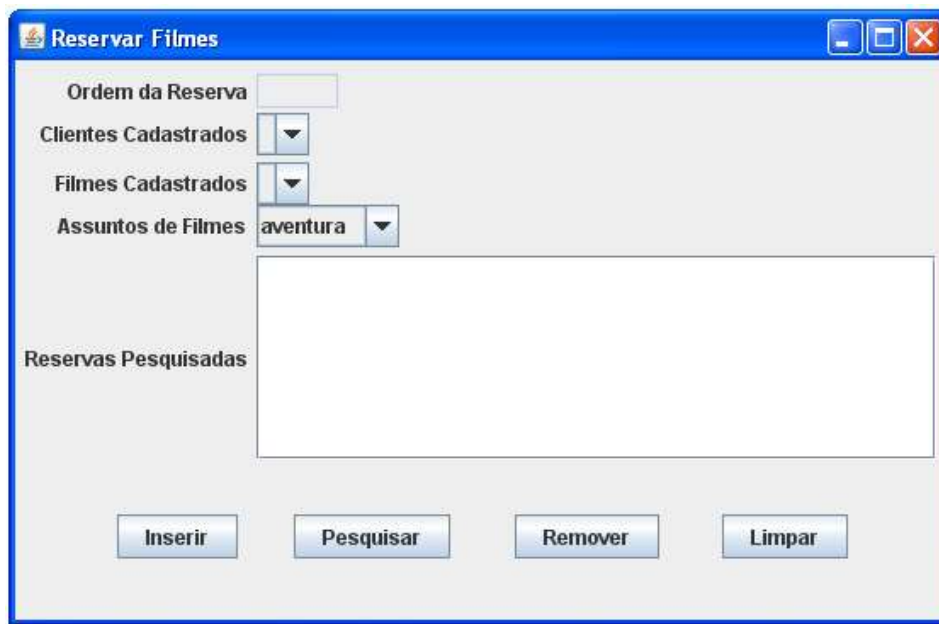
- ControladorReserva()
- String inserirReserva (String chave\_cliente, int chave\_filme)
- String removerReserva (String chave\_cliente, int chave\_filme)

## Exercícios

2.3 – Crie a classe interfaces.JanelaReserva, conforme a figura do slide seguinte:

- dados adicionais às variáveis do componentes da interface gráfica:
  - ControladorReserva controlador
  - Vector<Visão<String>> clientes\_cadastrados
  - Vector<Visão<Integer>> filmes\_cadastrados

## Exercícios



UFGD - POO-E4 - Joinville Batista Junior

71

## Exercícios

### 2.3.1 – Composição

- labels dos componentes: Label
- ordem da reserva: TextField
- clientes cadastrados, filmes cadastrados, assuntos de filmes: ComboBox
- reservas pesquisadas: ScrollPane encapsulando TextArea
- painel de comandos do Caso de Uso: Panel
  - vincular ao painel, os botões (Button) associados aos comandos do Caso de Uso

### 2.3.2 – Propriedades específicas

- basear-se no Caso de Uso Cadastrar Filmes

### 2.3.3 – Inicialização

- basear-se no Caso de Uso Cadastrar Clientes

UFGD - POO-E4 - Joinville Batista Junior

72

## Exercícios

### 2.3.4 – Eventos

- void inserirReserva (ActionEvent)
  - obter a visão do cliente selecionado e a visão do filme selecionado
  - solicitar ao controlador a inserção da reserva no BD, a partir das chaves do cliente e filme selecionados
  - informar o sequencial criado para a reserva pelo BD
  - limpar os campos que não estejam associados a criação da reserva

## Exercícios

- void pesquisarReservas(ActionEvent)
  - ler os filtros de reserva: chave de cliente e de filme (obtidas a partir das visões selecionadas), e assunto do filme
  - solicitar a pesquisa das reservas cadastradas que atendem os filtros selecionados
  - formatar os dados de cada reserva retornada e apendar à área de reservas pesquisadas

## Exercícios

- void removerReserva(ActionEvent)
  - obter a visão do cliente selecionado e a visão do filme selecionado
  - solicitar ao controlador a remoção da reserva no BD, a partir das chaves do cliente e filme selecionados

## Exercícios

### 3 – Implemente o Caso de Uso Cadastrar Cópias de Filmes

- inserção
  - o usuário deve selecionar um filme cadastrado e informar: data, valor pago e quantidade adquirida
- remoção
  - o usuário deve selecionar um filme cadastrado, uma cópia do filme e informar o motivo da remoção [perdida, danificada, vendida]
- novas classes de entidade
  - CópiaFilme: filme, sequencial, data de aquisição, valor de aquisição, estado [disponível, locada, reservada]

## Exercícios

### 4 – Implemente o Caso de Uso Locar Cópias de Filmes

- locação
  - o usuário deverá selecionar o cliente, selecionar as cópias de filmes e solicitar a locação
  - o sistema deverá verificar se o cliente está cadastrado e se as cópias de filmes estão disponíveis ou reservadas para o cliente
  - o sistema deverá informar:
    - nome do cliente
    - para cada filme locado: título do filme, data de devolução, e valor da locação
- novas classes de entidade
  - Categoria: categoria [catálogo, lançamento, promoção], valor da locação)
  - Locação: cliente, cópias de filmes, data

## Exercícios

### 4 – Implemente o Caso de Uso Devolver Cópias de Filmes:

- devolução
  - o usuário deverá selecionar o cliente, as cópias de filmes locadas a serem devolvidas e solicitar a devolução
  - o sistema deverá verificar se o cliente está cadastrado e se as cópias de filmes estão locadas para o cliente
  - o sistema deverá reservar a cópia de filme que for devolvida, para a qual exista uma reserva de filme, e remover a reserva
  - o sistema deverá informar
    - nome do cliente
    - para cada filme devolvido: título do filme, valor a ser pago
    - total a ser pago

## Exercícios

### 5 – Implemente o Caso de Uso Pesquisar Filmes:

- pesquisa
  - detalhar
- novas classes de entidade
  - Diretor: nome, filmes com oscar
  - Ator: nome, filmes com oscar, premiação [principal, coadjuvante]