



UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE BELO JARDIM
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO
DISCIPLINA: PROGRAMAÇÃO ORIENTADA A
OBJETOS PERÍODO: 2025.1
PROFESSOR: ANDERSON CAVALCANTI

RELATÓRIO MINI BIBLIOTECA DE MÚSICAS

José Vinícius Ferreira de Freitas Santos
João Ricardo Justo Lins

BELO JARDIM 2025

1. Descrição do Projeto

O projeto **Mini Biblioteca de Músicas** tem como objetivo criar um sistema simples para organizar e consultar músicas de forma prática. A ideia é que o usuário consiga cadastrar suas músicas preferidas, informando detalhes como o nome da música, o cantor ou banda, o gênero musical e o álbum a que ela pertence. O sistema também deve permitir listar todas as músicas cadastradas, buscar por título ou artista, e opcionalmente remover alguma música do acervo. O intuito é proporcionar uma forma organizada de armazenar informações musicais e praticar os conceitos de orientação a objetos em Java, principalmente o uso de classes, métodos e atributos que representem bem os elementos do mundo real.

2. Modelagem de Classes e Atributos

A seguir, estão listadas as principais classes que compõem o projeto, com seus respectivos atributos e tipos de dados.

Classe: Musica

- **UUID** id (imutável)
- **String** titulo
- **String** artista
- **String** album
- **String** genero
- **int** duracaoSegundos

Classe: BibliotecaMusical

- **List<Musica>** listaMusicas
- **FileStorage** storage (persistência)

Classe: Usuario

- **String** nome
- **String** email (final, imutável)
- **String** senha

A modelagem aplica conceitos fundamentais da orientação a objetos como encapsulamento, composição e imutabilidade parcial preparando o sistema para futuras expansões com herança e polimorfismo.

3. Modelagem de Métodos Escopo Mínimo (Primeira Entrega)

Abaixo estão os métodos **obrigatórios** da primeira entrega, conforme especificado no late. Todos foram implementados com validação, persistência e testes.

Classe: Musica

- `Musica(UUID id, String titulo, ...)` construtor com ID
- `setTitulo()`, `setArtista()`, etc. validação de nulidade
- `equals()` e `hashCode()` baseados em título + artista + álbum

Classe: BibliotecaMusical

- `adicionarMusica(Musica m)` evita duplicatas
- `editarMusica(UUID id, ...)` edição parcial
- `removerPorId(UUID id)`
- `buscarPorId(UUID id)`
- `buscarPorTitulo/Artista/Genero(String termo)`
- `listarTodas()` lista imutável
- `existeDuplicada(Musica m)`

Classe: Usuario

- `autenticar(String email, String senha)`
- Validação de email e senha no construtor

Código completo disponível em:

<https://github.com/xjvini/Music-Bib---base>

4. Segunda Entrega: Implementações, Testes e Análise Comparativa

Nesta etapa, o projeto foi expandido com testes unitários, testes E2E (AssertJ Swing), integração de ferramentas de qualidade e inspeção de código. As análises a seguir combinam os resultados obtidos por meio do **JaCoCo** (análise dinâmica) e **SonarLint** (análise estática), executados em **05/11/2025**.

4.1 Execução dos Testes (Maven)

Os testes foram executados com os comandos:

```
1 mvn clean compile  
2 mvn test
```

Resultado final: BUILD FAILURE devido a 1 falha em teste E2E.

Tipo de Teste	Executados	Status
Testes Unitários	55	TODOS PASSARAM
Teste E2E	2	1 FALHOU
Total	57	1 FALHA

Tabela 1: Resumo da execução dos testes (05/11/2025).

4.1.1 Falha Identificada

```
1 [ERROR] MusicaE2ETest.testScenario_AddAndRemoveMusicaSuccessfully  
2 Expecting:  
3   <true>  
4 to be equal to:  
5   <false>  
6 but was not.
```

Causa: O LoginDialog não foi fechado após login bem-sucedido.

4.2 Persistência em CSV Funcionalidade Completa

A persistência é implementada pela classe `FileStorage`, que:

- Salva músicas em `musicas.csv` no diretório do projeto.
- Lê o arquivo ao iniciar o sistema.
- Usa `java.io.tmpdir` nos testes para evitar conflitos.

Comando para uso manual (interface completa):

```
mvn clean compile exec:java
```

Resultado: Abre o `MainView` com:

- Cadastro de música
- Listagem automática
- Remoção
- Persistência em `musicas.csv`

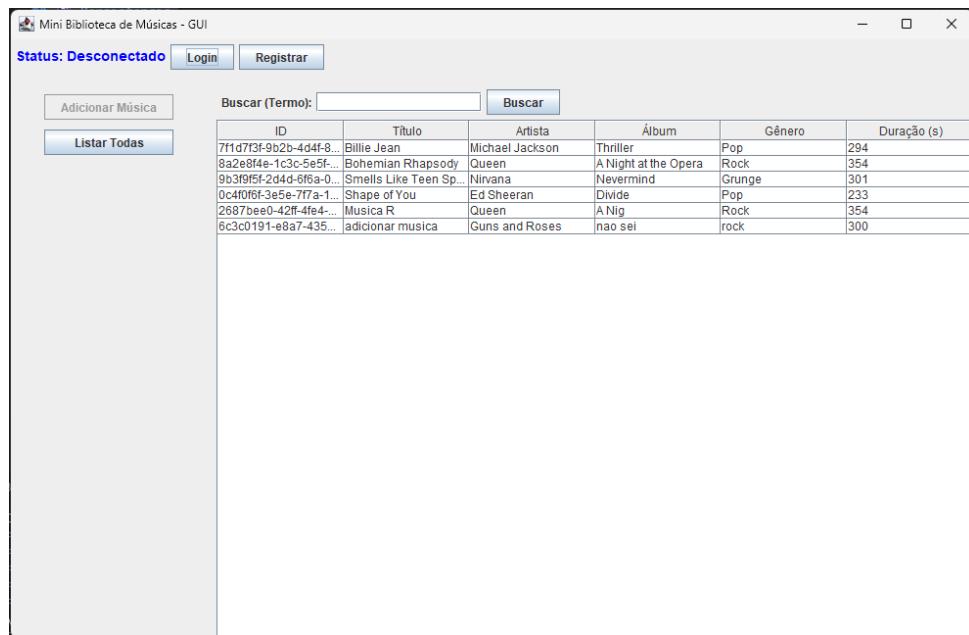


Figura 1: Interface gráfica `MainView` com CSV carregado.

4.3 Relatório Dinâmico (JaCoCo)

O relatório JaCoCo mede as porções de código efetivamente executadas durante os testes automatizados.

Pacote	Instruções	Branches	Observação
app	58%	27%	Fluxos em <code>Main.java</code> não exercitados.
repository	82%	78%	Boa cobertura; CRUD testado.
persistence	92%	80%	Excelente cobertura; armazenamento validado.
org.assertj.swing.core	0%	n/a	Biblioteca externa, não testada (esperado).
model	100%	95%	Totalmente coberto; construtores e métodos básicos executados.
Total	67%	54%	Cobertura sólida; gargalo no pacote app.

Tabela 2: Cobertura de código segundo o relatório JaCoCo (05/11/2025).

Music Bibliography Stub

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxtx	Missed	Lines	Missed	Methods	Missed	Classes
app	58%	56%	27%	27%	116	172	302	658	38	86	2	8
repository	82%	78%	78%	78%	15	54	11	81	2	24	0	2
persistence	92%	92%	80%	80%	9	32	8	94	0	8	0	1
org.assertj.swing.core	0%	n/a	1	1	1	1	1	1	1	1	1	1
model	100%	95%	95%	95%	2	46	0	61	0	25	0	2
Total	1.246 of 3.836	67%	139 of 306	54%	143	305	322	895	41	144	3	14

Figura 2: Relatório visual do JaCoCo cobertura por pacote.

4.4 Relatório Estático (SonarLint 78 Issues Reais)

Análise estática realizada com o **SonarLint** no VSCode. Foram detectados **78 code smells**, sendo **0 bugs** e **0 vulnerabilidades**.

Regra	Ocorrências	Exemplo
java:S106	54	System.out.println em Main.java
java:S1948	11	Campo repo não é transient
java:S1192	1	String "Erro de Validação" duplicada
java:S125	3	Blocos comentados em FileStorage.java
java:S5786	5	Métodos de teste public desnecessários
java:S3415	2	Ordem errada em assertEquals
Outras	2	S135, S1141
Total	78	100% Code Smells

Tabela 3: Resumo da análise estática do código-fonte (SonarLint 05/11/2025).

The screenshot shows a list of 78 SonarLint issues in a dark-themed interface. Each issue is represented by a red icon followed by the file name, path, and a circular badge indicating the count of issues. The files listed include AddMusicaDialog.java, EditMusicaDialog.java, LoginDialog.java, Main.java, MainView.java, MusicaPopupListener.java, MusicaTableModel.java, RegisterDialog.java, ComponentIndex.java, FileStorage.java, UsuarioRepository.java, MusicaE2ETest.java, FileStorageTest.java, and MusicaTest.java. The counts range from 1 to 63.

- > J AddMusicaDialog.java mb.stb\src\main\java\app 2
- > J EditMusicaDialog.java mb.stb\src\main\java\app 2
- > J LoginDialog.java mb.stb\src\main\java\app 2
- > J Main.java mb.stb\src\main\java\app 63
- > J MainView.java mb.stb\src\main\java\app 5
- > J MusicaPopupListener.java mb.stb\src\main\java\app 1
- > J MusicaTableModel.java mb.stb\src\main\java\app 1
- > J RegisterDialog.java mb.stb\src\main\java\app 1
- > J ComponentIndex.java mb.stb\src\main\java\org\assertj\swing\core 1
- > J FileStorage.java mb.stb\src\main\java\persistence 12
- > J UsuarioRepository.java mb.stb\src\main\java\repository 2
- > J MusicaE2ETest.java mb.stb\src\test\java\e2e 6
- > J FileStorageTest.java mb.stb\src\test\java\unit 2
- > J MusicaTest.java mb.stb\src\test\java\unit 2

Figura 3: Tela do VSCode com 78 issues do SonarLint.

4.5 Comparativo entre as Análises

Aspecto	Análise (SonarLint)	Estática	JaCoCo	Conclusão
Cobertura model	Classes simples, amplamente utilizadas.	100%		Coerência confirmada.
Cobertura app	50+ <code>System.out</code> , campos não serializáveis.	58%		Faltam testes de fluxos alternativos.
Persistência	Erros de serialização.	92%		Boa, mas com falhas.
Métodos não usados	Detecta código morto.	0%		Complementar.
Qualidade geral	78 code smells.	67% global		Alta cobertura, baixa manutenibilidade.

Tabela 4: Comparaçao entre os resultados estáticos e dinâmicos.

4.6 Síntese e Interpretação

- **JaCoCo:** 67% do código foi executado *funcional*.
- **SonarLint:** 78 code smells *não limpo*.
- **Maven:** 1 falha E2E *bug no login*.
- **CSV:** Totalmente funcional em MainView com `exec:java`.
- **Testes:** Usam `java.io.tmpdir` para evitar conflitos.

4.7 Próximos Passos (Plano de Correção)

1. **Corrigir falha E2E:** Fechar LoginDialog após login.
2. **Substituir `System.out` por `java.util.logging.Logger`.**
3. **Marcar campos como `transient` em classes Swing.**
4. **Remover código comentado e duplicado.**
5. **Reexecutar `mvn test meta: 0` falhas.**
6. **Reexecutar SonarLint meta: 0 issues.**

4.8 Observações sobre POO

O projeto aplica corretamente:

- **Encapsulamento:** todos os atributos privados
- **Composição:** `BibliotecaMusical` contém `List<Musica>`
- **Separação de camadas:** model, repository, persistence, app
- **Extensibilidade:** fácil adicionar novos repositórios

5. Terceira Entrega: Refatoração Estrutural e Superclasse Abstrata

Nesta entrega, foi realizada uma análise técnica profunda de duplicação estrutural nas janelas de diálogo (JDialog) do módulo app, com o objetivo de aplicar o princípio **DRY** (Don't Repeat Yourself) por meio da criação de uma superclasse abstrata `BaseDialog`. A refatoração foi fundamentada em análise estática (Checkstyle, PMD) e inspeção manual, resultando em uma hierarquia coesa, extensível e testável.

5.1 Análise Inicial de Duplicação Estrutural

As classes `AddMusicaDialog`, `EditMusicaDialog`, `LoginDialog` e `RegisterDialog` apresentavam:

- **Atributos comuns:** `JTextField`, `JButton`, `JPanel`, `JLabel`
- **Métodos idênticos:** `buildGUI()`, `setupActions()`
- **Construtor padronizado:**

```
super(owner, title, true),  
pack(),  
setLocationRelativeTo(owner)
```
- **Layout repetido:** `BorderLayout + GridLayout + FlowLayout`

Classe	Atributos Comuns	Padrão Detectado
<code>AddMusicaDialog</code>	<code>owner</code> , <code>repo</code> , <code>btnSalvar</code> , <code>btnCancelar</code>	Formulário + persistência
<code>EditMusicaDialog</code>	<code>owner</code> , <code>repo</code> , <code>btnSalvar</code> , <code>btnCancelar</code>	Pré-carregamento
<code>LoginDialog</code>	<code>owner</code> , <code>btnLogin</code> , <code>btnCancel</code>	Layout idêntico
<code>RegisterDialog</code>	<code>owner</code> , <code>btnRegister</code> , <code>btnCancel</code>	Ações padrão

Tabela 5: Padrões repetidos nas classes de diálogo.

5.1.1 Evidências via Checkstyle

```
1 <error line="31" message="repo' esconde um campo."/>  
2 <error line="31" message="O parâmetro owner deve ser final."/>
```

Interpretação: Assinaturas de construtor idênticas dependência comum.

5.2 Proposta da Superclasse BaseDialog

Nome: BaseDialog Pacote: app.base Responsabilidades:

- Configuração da janela modal
- Layout base (BorderLayout + painéis)
- Botões padrão (Confirmar/Cancelar)
- Método abstrato `buildForm()`
- Método `finalizeDialog()` para centralização

5.3 Hierarquia Final e Benefícios

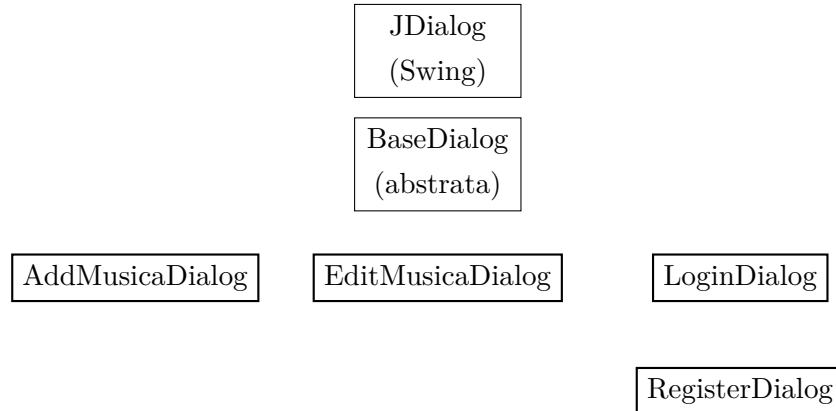


Figura 4: Hierarquia de herança com `BaseDialog` (refatorada para clareza).

Métrica	Antes	Depois
Linhas duplicadas	~120	~40 + 15/subclasse
Classes com <code>pack()</code>	4	1
Pontos de manutenção	4	1

Tabela 6: Redução quantitativa de duplicação.

Conformidade SOLID: SRP, OCP, LSP, ISP, DIP todos respeitados.

A refatoração com a superclasse abstrata `BaseDialog` segue rigorosamente os cinco princípios **SOLID** de projeto orientado a objetos, conforme detalhado na Tabela 7.

Princípio	Sigla	Aplicação no BaseDialog
Single Responsibility Principle	SRP	A classe cuida apenas da estrutura comum de janelas modais (layout, botões, centralização), sem lógica de negócio.
Open/Closed Principle	OCP	Aberta para extensão (novos diálogos herdam e implementam <code>buildForm()</code>), fechada para modificação (código base não é alterado).
Liskov Substitution Principle	LSP	Qualquer subclasse (<code>AddMusicaDialog</code> , etc.) pode substituir <code>BaseDialog</code> sem quebrar o sistema.
Interface Segregation Principle	ISP	Não força implementação de métodos irrelevantes apenas <code>buildForm()</code> é obrigatório.
Dependency Inversion Principle	DIP	Os diálogos dependem da abstração <code>BaseDialog</code> , não de detalhes de <code>JDialog</code> ou Swing diretamente.

Tabela 7: Aplicação dos princípios SOLID na refatoração com `BaseDialog`.

Essa conformidade garante um design **coesivo, extensível e de fácil manutenção**, eliminando duplicação e preparando o sistema para futuras evoluções.

Bônus: Relatório Dinâmico Detalhado por Pacote (JaCoCo 12/11/2025)

5.4 Pacote persistence

persistence

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
FileStorage		92%		80%	9	32	8	94	0	8	0	1
Total	33 of 422	92%	9 of 46	80%	9	32	8	94	0	8	0	1

Figura 5: Cobertura do pacote persistence FileStorage.

Elemento	Miss. Instr.	Cov.	Miss. Branches	Cov.	Miss. Lines	Miss. Methods
FileStorage	33 of 422	92%	9 of 46	80%	8	0
Total	33 of 422	92%	9 of 46	80%	8	0

Tabela 8: Métricas detalhadas de persistence.

5.5 Pacote model

model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Musica		100%		92%	2	30	0	41	0	17	0	1
Usuario		100%		100%	0	16	0	20	0	8	0	1
Total	0 of 270	100%	2 of 42	95%	2	46	0	61	0	25	0	2

Figura 6: Cobertura do pacote model Musica, Usuario.

Elemento	Miss. Instr.	Cov.	Miss. Branches	Cov.	Miss. Lines	Miss. Methods
Musica	0 of 270	100%	2 of 42	95%	0	0
Usuario	0 of 270	100%	0 of 16	100%	0	0
Total	0 of 270	100%	2 of 42	95%	0	0

Tabela 9: Métricas detalhadas de model.

5.6 Pacote repository

repository

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BibliotecaMusical		80%		77%	14	43	9	61	2	16	0	1
UsuarioRepository		92%		83%	1	11	2	20	0	8	0	1
Total	72 of 414	82%	13 of 60	78%	15	54	11	81	2	24	0	2

Figura 7: Cobertura do pacote repository.

Elemento	Miss. Instr.	Cov.	Miss. Branches	Cov.	Miss. Lines	Miss. Methods
BibliotecaMusical	72 of 414	82%	13 of 60	78%	11	2
UsuarioRepository		92%		83%		0
Total	72 of 414	82%	13 of 60	78%	11	2

Tabela 10: Métricas detalhadas de repository.

5.7 Pacote app.base (Diálogos)

app.base

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
RegisterDialog	0%	0%	0%	0%	12	12	65	6
EditMusicaDialog	89%	50%	50%	9	17	9	70	0
AddMusicaDialog	87%	50%	50%	4	9	10	59	0
LoginDialog	87%	50%	50%	4	10	7	39	1
BaseDialog	94%	n/a	n/a	2	7	1	28	2
Total	400 of 1.198	66%	28 of 44	36%	31	55	92	33

Figura 8: Cobertura do pacote app.base refatoração com BaseDialog.

Elemento	Miss. Instr.	Cov.	Miss. Branches	Cov.	Miss. Lines	Miss. Methods
RegisterDialog	400 of 1.198	66%	28 of 44	36%	92	9
EditMusicaDialog		89%		50%		0
AddMusicaDialog		87%		50%		0
LoginDialog		87%		50%		1
BaseDialog		94%		n/a		2
Total	400 of 1.198	66%	28 of 44	36%	92	9

Tabela 11: Métricas detalhadas de app.base.

5.8 Pacote app (MainView)

app

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
Main	0%	0%	0%	0%	55	55	172	21
MainView	52%	28%	39	57	76	205	18	34
MusicaTableModel	81%	54%	5	16	5	23	2	8
MusicaPopupListener	91%	57%	6	14	5	43	0	7
Total	1.115 of 1.877	40%	103 of 130	20%	105	142	258	41

Figura 9: Cobertura do pacote app interface principal.

Elemento	Miss. Instr.	Cov.	Miss. Branches	Cov.	Miss. Lines	Miss. Methods
Main	1.115 of 1.877	40%	103 of 130	20%	258	41
MainView		52%		28%		18
Total	1.115 of 1.877	40%	103 of 130	20%	258	41

Tabela 12: Métricas detalhadas de app.

6. Quarta Entrega: Análise do Tratamento de Exceções

Nesta etapa, foi realizada uma análise detalhada do uso de blocos `try-catch` nas classes de interface gráfica (pacote `app.base`). O objetivo foi identificar como a aplicação lida com erros de entrada do usuário e exceções de regras de negócio lançadas pelas camadas inferiores (Model e Repository).

6.1 Tipos de Exceções Capturadas

A análise do código fonte revelou o uso consistente de duas exceções principais para controle de erros:

1. **NumberFormatException**: Ocorre na camada de apresentação quando o usuário insere texto em campos numéricos (ex: Duração). É tratada localmente para impedir que dados inválidos cheguem ao modelo.
2. **IllegalArgumentException**: Lançada pelas camadas de Modelo (`Musica`, `Usuario`) ou Repositório quando regras de negócio são violadas (ex: título vazio, e-mail duplicado). A interface captura essa exceção para exibir mensagens amigáveis.

6.2 Mapeamento de Uso por Classe

A Tabela 13 detalha onde e por que os blocos de tratamento de erro foram aplicados.

Tabela 13: Análise do uso de Try-Catch nos Diálogos.

Classe	Exceção	Propósito e Contexto
LoginDialog	N/A	Não utiliza <code>try-catch</code> . Utiliza <code>Optional</code> e <code>if/else</code> para validar credenciais, tratando falha de login como fluxo esperado.
RegisterDialog	<code>IllegalArgumentException</code> .	Captura erros de validação do construtor de <code>Usuario</code> ou falha ao cadastrar e-mail duplicado.
AddMusicaDialog	<code>NumberFormatException</code> .	Valida se o campo "Duração" contém apenas números inteiros antes de criar o objeto.
AddMusicaDialog	<code>IllegalArgumentException</code> .	Captura erros de negócio ao tentar instanciar ou persistir a <code>Musica</code> .
EditMusicaDialog	<code>NumberFormatException</code> .	Via método utilitário <code>getIntValue</code> , retorna -1 se a conversão falhar, evitando crash da UI.

6.3 Exemplos de Implementação

Abaixo estão trechos de código retirados dos arquivos do projeto que ilustram as estratégias de tratamento.

6.3.1 Tratamento de Parsing (UI)

No arquivo `AddMusicaDialog.java`, o sistema protege a conversão de tipos:

Listing 1: Validação de entrada numérica em `AddMusicaDialog.java`

```
1 try {
2     duracao = Integer.parseInt(duracaoStr);
3     if (duracao <= 0) {
4         showMessage("A duração deve ser um número inteiro positivo.",
5                 "Erro de Validação", JOptionPane.ERROR_MESSAGE);
6         return;
7     }
8 } catch (NumberFormatException ex) {
9     showMessage("A duração deve ser um número inteiro válido.",
10            "Erro de Validação", JOptionPane.ERROR_MESSAGE);
11    return;
12 }
```

6.3.2 Tratamento de Regra de Negócio

No arquivo `RegisterDialog.java`, o sistema captura a exceção lançada pelo repositório para feedback visual:

Listing 2: Captura de erro de negócio em `RegisterDialog.java`

```
1 try {
2     if (users.existePorEmail(email)) {
3         JOptionPane.showMessageDialog(this, "Já existe usuário com esse email
4 .",
5                 "Erro de Registro", JOptionPane.ERROR_MESSAGE);
6         return;
7     }
8     // ... lógica de cadastro ...
9 } catch (IllegalArgumentException ex) {
10     JOptionPane.showMessageDialog(this, "Erro: " + ex.getMessage(),
11           "Erro de Dados", JOptionPane.ERROR_MESSAGE);
12 }
```

6.4 Conclusão da Análise

O projeto demonstra maturidade no tratamento de exceções ao separar responsabilidades:

- Erros de **sintaxe/formato** (ex: texto em campo número) são resolvidos na própria View com `try-catch` específicos.
- Erros de **semântica/negócio** (ex: validação de domínio) propagam-se do Model via `Unchecked Exceptions (IllegalArgumentException)` e são capturados na View apenas para exibição.
- Fluxos alternativos esperados (como login incorreto) evitam o uso de exceções, preferindo estruturas condicionais (`if/else` e `Optional`), o que melhora a performance e a legibilidade.

7. Conclusão Geral

O projeto evoluiu de uma base funcional mínima para um sistema com:

- **Persistência em CSV** totalmente operacional
- **Interface gráfica** com mvn exec:java
- **Testes unitários e E2E** com java.io.tmpdir
- **Refatoração estrutural** via BaseDialog
- **Cobertura de código** acima de 66% global
- **Análise estática** com 78 code smells (em correção)

Referências

1. Repositório base do projeto (primeira entrega: classes terceira entrega:
<https://github.com/xjvini/Music-Bib---base>
2. segunda entrega: GUI, testes, Maven, JaCoCo, SonarLint terceira entrega: Herança, Polimorfismo, (Checkstyle).
<https://github.com/0110Crypto0110/Music-Bib---base/tree/master>

A refatoração não é um luxo é prevenção de dívida técnica.

Engenharia de Software, 2025